Automatisation du test Tous-Les-Chemins en présence d'appels de fonction

Soutenance de thèse de doctorat

Patricia Mouy

CEA/Saclay-LIST Laboratoire de Sûreté de Logiciels
Université d'Evry Val d'Essonne

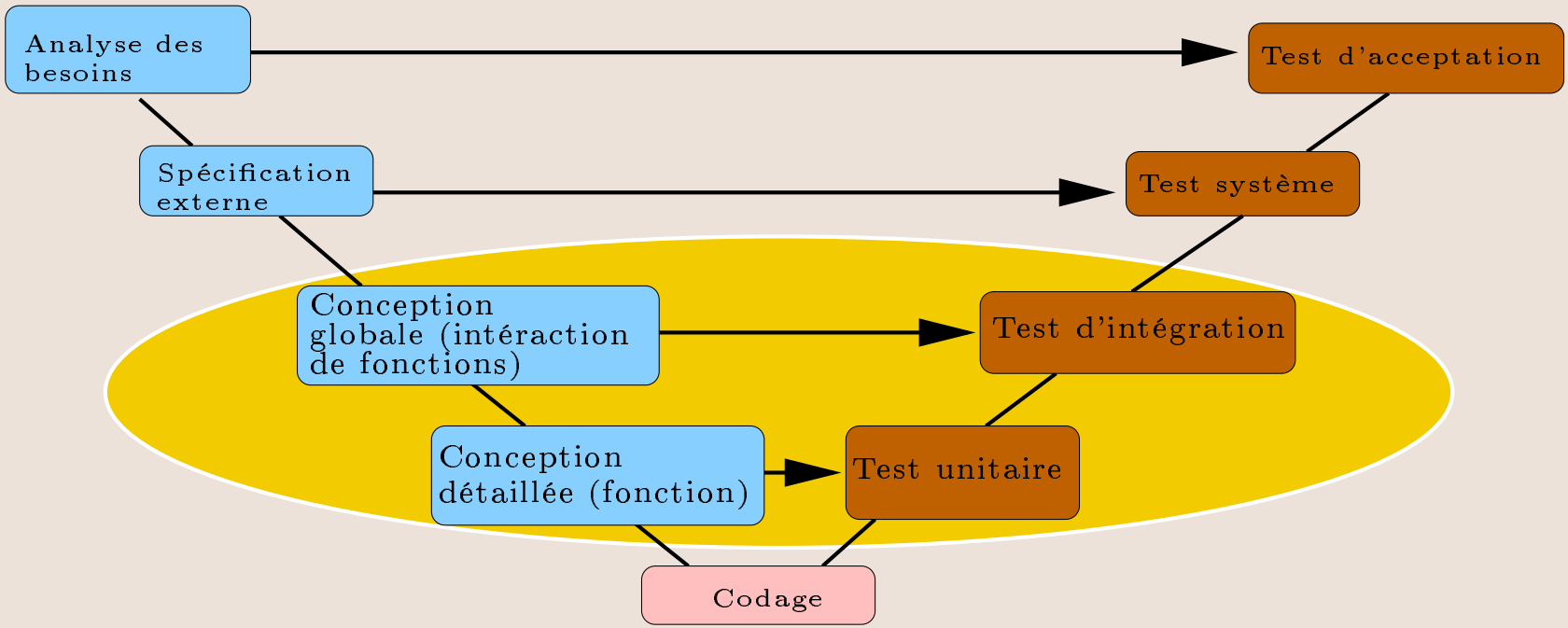
16 mai 2007

Test

Objectif du test :

«Tester c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts. » [The Art of Software Testing, Myers]

Premières étapes de test

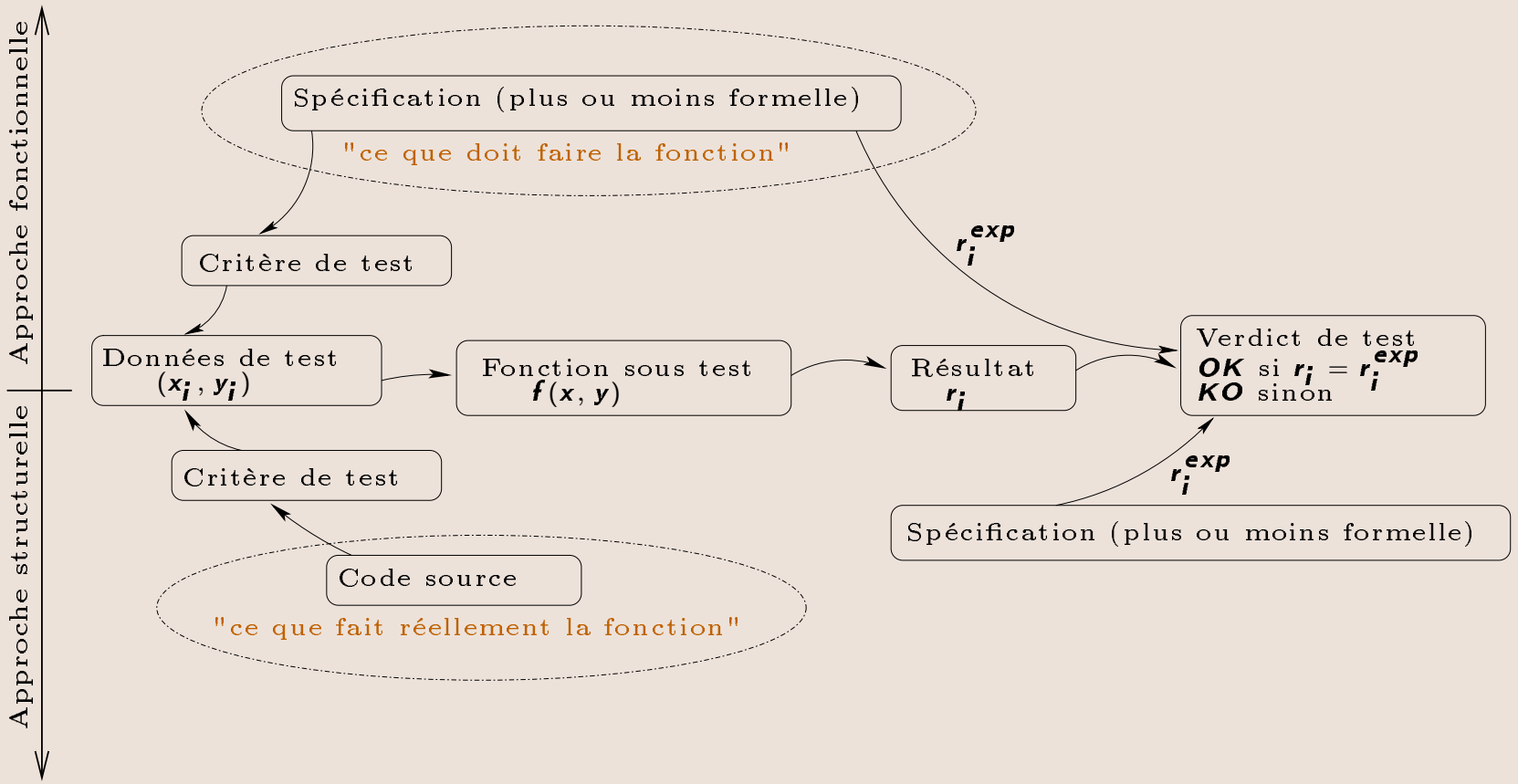


Test unitaire

Test unitaire

Test isolé d'une fonction i.e. indépendamment du système auquel elle appartient.

Complémentarité des approches fonctionnelle et structurelle

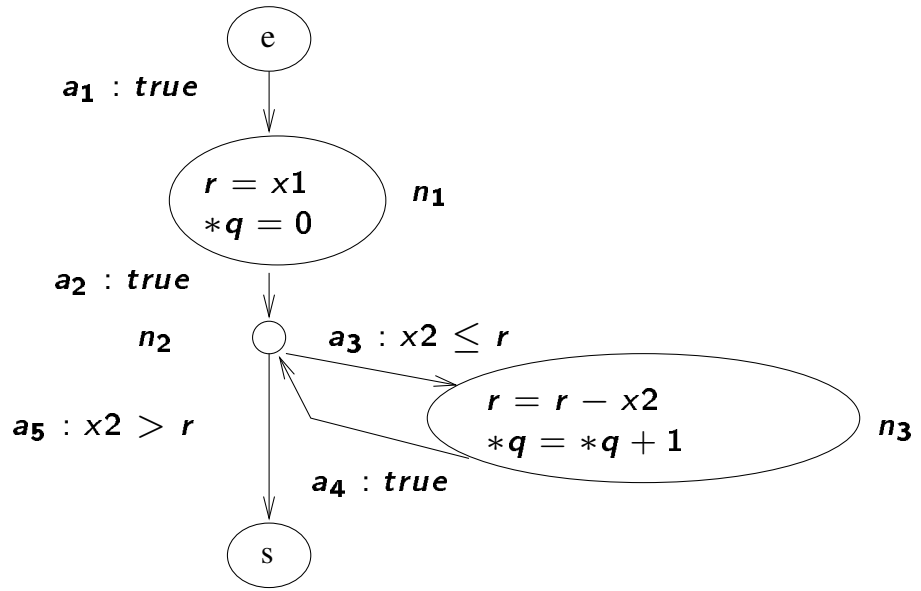


Test unitaire structurel Tous-Les-(k)-Chemins

Code source

```
int div(int x1, int x2, int *q)
{
int r=x1;
*q=0;
while (x2<=r)
{
  r=r−x2;
  *q=*q+1;
}
return(r);
}
```

Graphe de flot de controle (CFG)



Données sélectionnées manuellement

Critère Tous-Les-k-Chemins pour $k = 2$:
Pas de passage :
$Ch_1 : x1 = 12, x2 = 18$
1 passage :
$Ch_2 : x1 = 10, x2 = 6$
2 passages :
$Ch_3 : x1 = 15, x2 = 7$

Critère Tous-Les-Chemins :
$Ch_1 : (e, a_1, n_1, a_2, n_2, a_5, s)$
$Ch_2 : (e, a_1, n_1, a_2, n_2, a_3, n_3, a_4, n_2, a_5, s)$
$Ch_3 : (e, a_1, n_1, a_2, n_2, a_3, n_3, a_4, n_2, a_3, n_3, a_4, n_2, a_5, s)$
$Ch_4 : (e, a_1, n_1, a_2, n_2, a_3, n_3, a_4, n_2, a_3, n_3, a_4, n_2, a_3, n_3, a_4, n_2, a_5, s)$
....

Problème

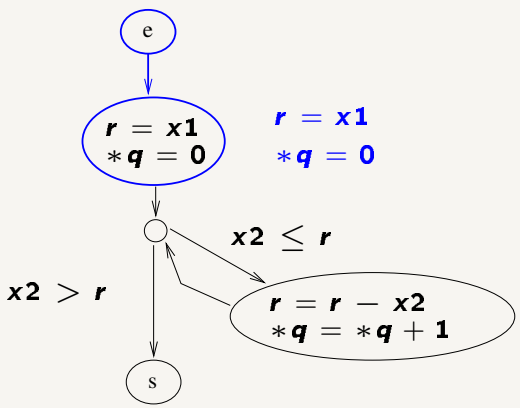
Comment déterminer automatiquement des données de test ?

Test unitaire : prédicat de chemin

$PC(f, Ch, X)$: prédicat du chemin Ch avec X vecteur des n variables d'entrée de la fonction f

Ensemble des conditions sur X pour l'exécution du chemin Ch de la fonction f

Prédicat de chemin $PC(div, Ch_2, (x1, x2))$ (1 passage dans la boucle)

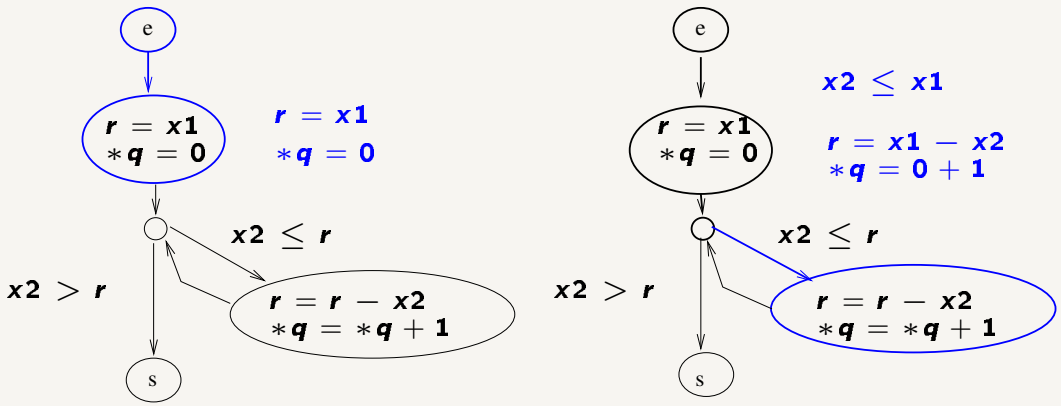


Test unitaire : prédicat de chemin

$PC(f, Ch, X)$: prédicat du chemin Ch avec X vecteur des n variables d'entrée de la fonction f

Ensemble des conditions sur X pour l'exécution du chemin Ch de la fonction f

Prédicat de chemin $PC(div, Ch_2, (x1, x2))$ (1 passage dans la boucle)

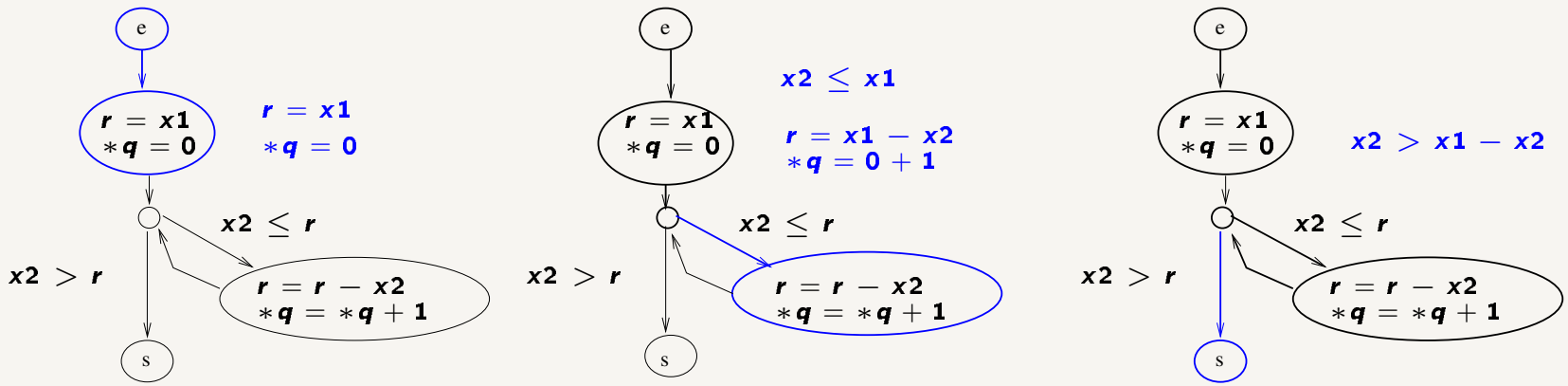


Test unitaire : prédicat de chemin

$PC(f, Ch, X)$: prédicat du chemin Ch avec X vecteur des n variables d'entrée de la fonction f

Ensemble des conditions sur X pour l'exécution du chemin Ch de la fonction f

Prédicat de chemin $PC(div, Ch_2, (x1, x2))$ (1 passage dans la boucle)



$PC(div, Ch_2, (x1, x2)) = (x2 \leq x1) \wedge (x2 > x1 - x2)$ soit après simplification
$PC(div, Ch_2, (x1, x2)) = (x2 \leq x1) \wedge (2 * x2 > x1)$
Données de test : $(x1 = 10, x2 = 6)$

Test unitaire : prédicat de chemin

Détermination des données de test

Donnée de test d'un chemin Ch = solution du prédicat de chemin $PC(f, Ch, X)$

Prédicat insatisfiable

- Prédicat de chemin insatisfiable $\Leftrightarrow$ chemin infaisable
- Indécidable en général et NP-complet sur domaines finis et chemins finis

Générer des données de test pour le critère Tous-Les-(k)-Chemins

- Identifier les chemins à couvrir
- Pour chaque chemin, pouvoir décider de l'insatisfiabilité du prédicat associé :
  - si satisfiable $\Rightarrow$ donnée de test
  - si insatisfiable $\Rightarrow$ choix d'un autre chemin à couvrir
  - si indéfini (TimeOut) $\Rightarrow$ choix d'un autre chemin à couvrir

Gestion de la présence d'appels de fonctions

Fonction sous test avec instruction d'appel pour la fonction div

```
int est_divisible(int x, int y)
{
  int res;
  int reste;
  int quo;
  if (x<0)
   x=-x;
  if (y<0)
   y=-y;

  reste=div(x,y,&quo) ;

  if (reste==0)
     res=1;
  else
    res=0;
  return(res);
}
```

Objectif

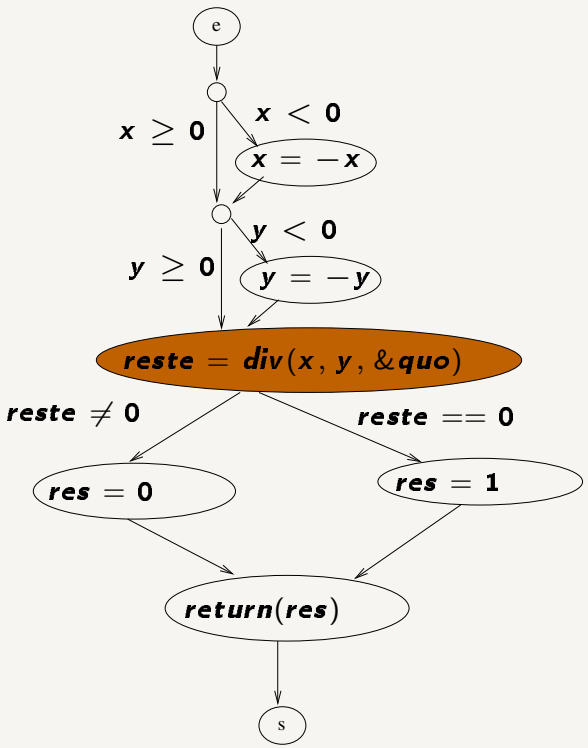
- Couverture de tous les chemins de la fonction `est_divisible`
- Prise en compte de l'instruction d'appel dans le calcul des prédicats de chemins

Pistes

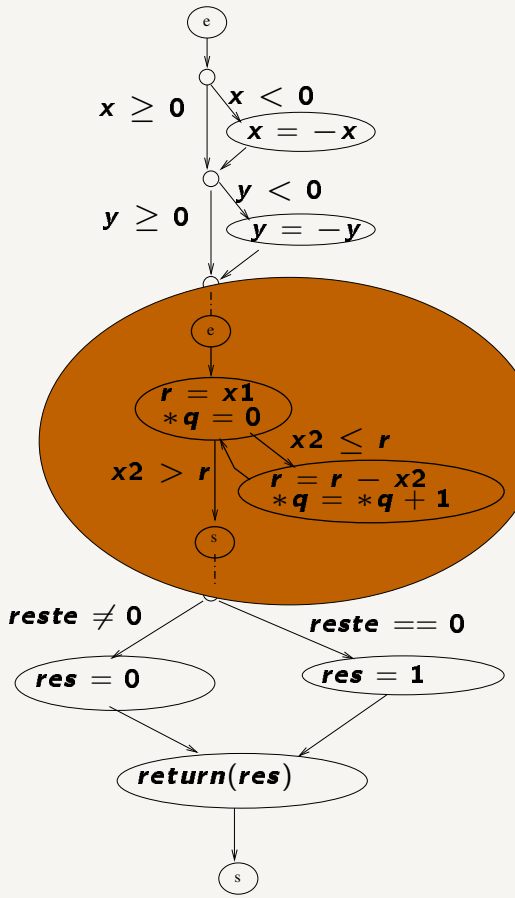
- Peu de travaux existants
- Extension du critère de test à la fonction appelée
  $\Rightarrow$ traitement "inlining"
- Substitution de la fonction appelée
  $\Rightarrow$ utilisation de bouchons

Approche par extension du critère à la fonction appelée : traitement "inlining"

CFG de la fonction sous test (appelante)



CFG déplié de la fonction sous test



Limitation

Prise en compte de la combinatoire de la fonction appelée

Approche par substitution : le cas des bouchons

Fonction sous test avec instruction d'appel pour la fonction div

```
int est_divisible(int x, int y)
{
  int res;
  int reste;
  int quo;
  if (x<0)
   x=-x;
  if (y<0)
   y=-y;

  reste=div(x,y,&quo) ;

  if (reste==0)
     res=1;
  else
    res=0;
  return(res);
}
```

Premier type de bouchons

Retourner toujours les mêmes valeurs

Illustration

Appel de `div` remplacé par `bStruc()` retournant la valeur 1

Limitations

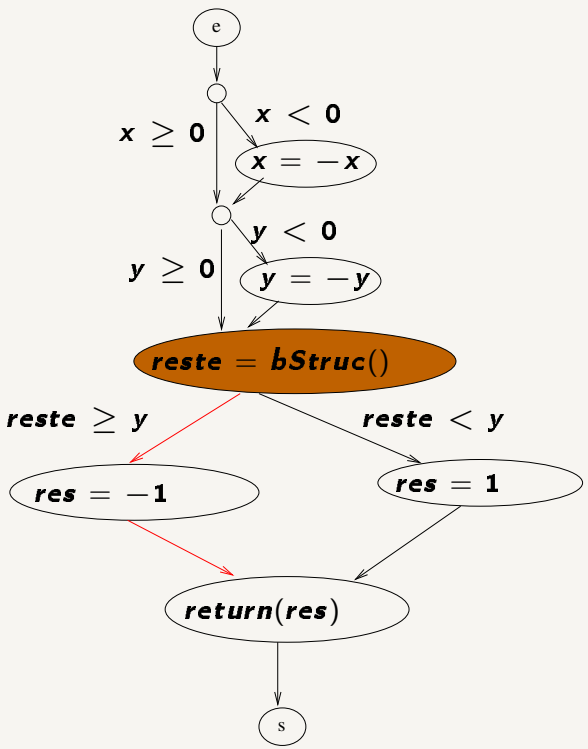
- Perte de couverture de la fonction sous test
- Comportement de la fonction appelée ignoré (verdict de test non fiable)

N.B. :

Utilisé généralement pour les appels système (`open()`,...)

Approche par substitution : le cas des bouchons

CFG de la fonction sous test (appelante)



Second type de bouchons

Retourner "toutes" les valeurs de la fonction appelée pour atteindre la couverture structurelle totale de la fonction sous test

Illustration

Appel de `div` remplacé par 2 bouchons `bStruc()` retournant :

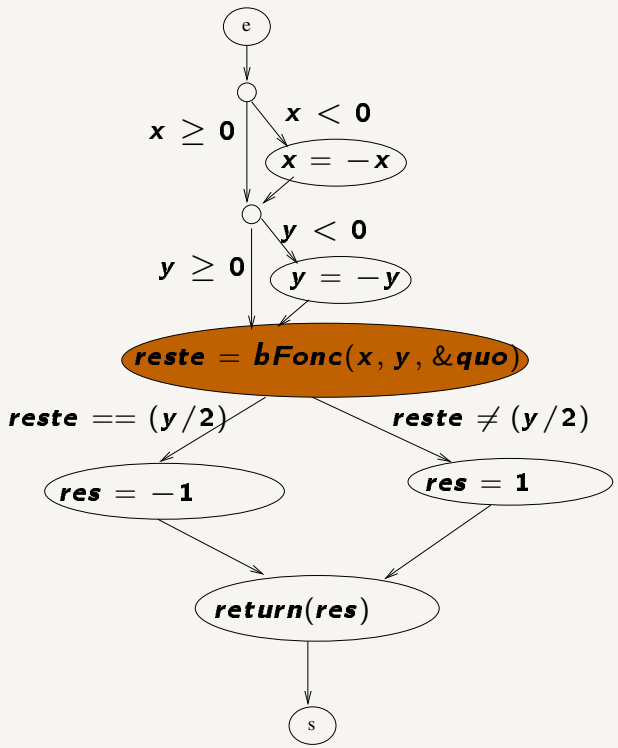
- une valeur $< y$ (branche "then")
- une valeur $\geq y$ (branche "else")

Limitations

- Comportement de la fonction appelée ignoré (verdict de test non fiable)
- Bouchon créé au cas par cas
- Couverture possible de chemins infaisables (chemin rouge)

Approche par substitution : le cas des bouchons

CFG de la fonction sous test (appelante)



Troisième type de bouchons

Simuler le comportement de la fonction appelée

Illustration

Appel remplacé par `bFonc(x,y,&quo)` retournant une valeur selon la spécification de la fonction appelée

Problèmes

Besoin de piloter les valeurs en sortie pour maintenir la couverture :

- table de correspondance au cas par cas et/ou incomplète,
- table de correspondance exhaustive sur les entrées (irréalisable)

Traitements habituels : bilan

Gestion des appels de fonction

- Cas particulier du test d'intégration
- Méthodes habituelles partielles et insuffisantes

Synthèse des limitations

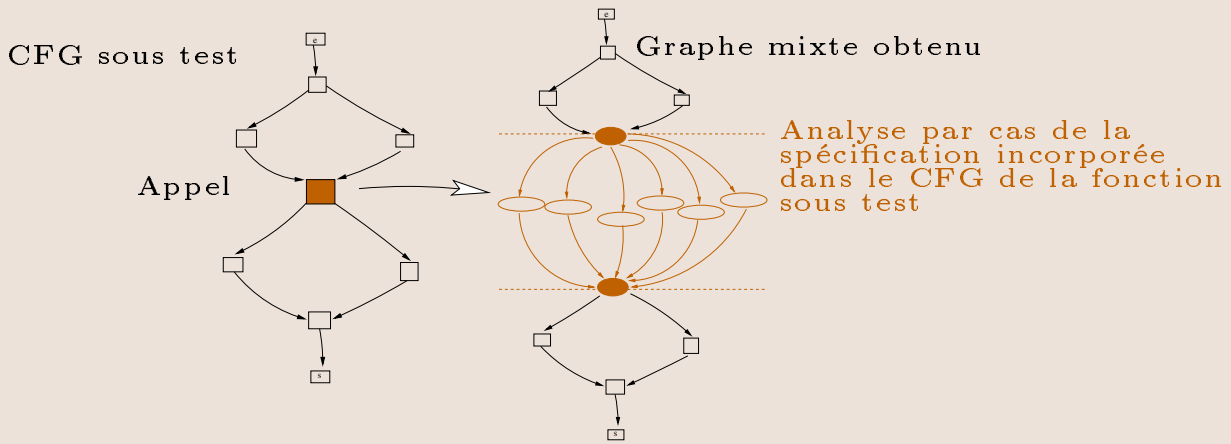
- Application impossible du critère Tous-Les-Chemins
- Pratiques manuelles lourdes
- Perte de couverture de la fonction sous test
- Couverture de chemins infaisables

Idée

Objectifs

- Maintien de la couverture de la fonction sous test
- Limiter l'exploration des fonctions appelées
- Automatiser la génération de cas de test en présence d'instructions d'appel

Comment ?



- Exploiter la spécification des fonctions appelées
- Graphe mixte : informations structurelles et informations fonctionnelles
- Extension de l'outil PathCrawler pour la gestion des appels

Plan

Langage de spécification utilisé

Langage pre/post

- Données : $W = (w_1, .., w_n)$ vecteur des n valeurs en entrée et $Z = (z_1, .., z_p)$ vecteur des p valeurs en sortie tels que $g(W) = Z$
- Couples pre/post : compromis entre expressivité et utilisabilité
- $Spec(g) : \{PP_i(W, Z, g)\}_{i \in [0..n]}$ avec $PP_i(W, Z, g) = (P_i(W), Q_i(W, Z))$:
  - $P_i(W)$ conditions sur W déterminant un sous-domaine en entrée pour g
  - $Q_i(W, Z)$ la relation entrées/sorties attendue pour le sous-domaine associé

Exemple pour la fonction de valeur absolue

- $W = (w_1)$, $Z = (z_1)$
- $Spec(abs, W, Z) = \{(w_1 < 0, z_1 = -w_1), (w1 \geq 0, z_1 = w_1)\}$

Exemple pour la fonction de division euclidienne

- $W = (w_1, w_2)$ avec w_1 : numérateur, w_2 : dénominateur et $Z = (z_1, z_2)$ avec z_1 : quotient et z_2 : reste
- $Spec(div, W, Z) : \{(w_1 \geq 0 \wedge w_2 > 0, w_1 = w_2 \times z_1 + z_2 \wedge 0 \leq z_2 \wedge z_2 < w_2)\}$

Langage de spécification utilisé

Caractéristiques

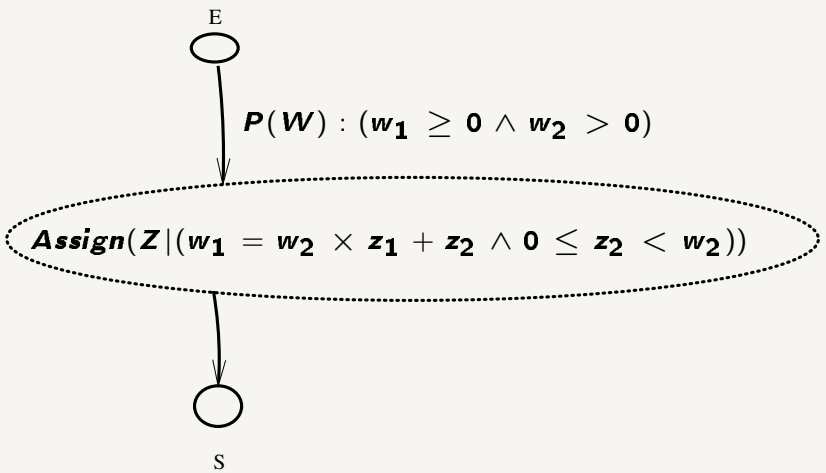
- Langage d'annotation pour langage C
- Logique du premier ordre
- Domaines finis (domaines des types C)
- Prédicats et opérateurs supplémentaires (quantificateurs, ...)
- Extension du langage possible avec expertise utilisateur

Restrictions imposées

- Pas de récursivité
- Déterminisme
- Complétude
- Couples pre/post à domaines disjoints

Graphe abstrait des fonctions appelées

Graphe abstrait de la fonction div



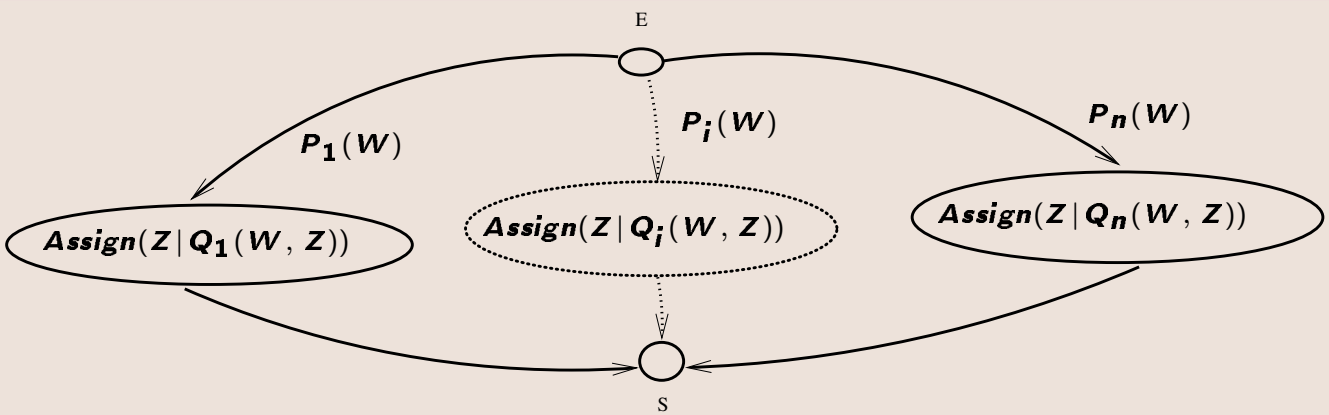
Graphe abstrait pour $Spec(g, W, Z)$ à n couples pre/post

Graphe connexe orienté à $2 + n$ nœuds et $2 * n$ arcs tels que :

- n arcs étiquetés des conditions $P_i(W)$ et
- n nœuds étiquetés des prédicats existentiels $Assign(Z|Q_i(W, Z))$

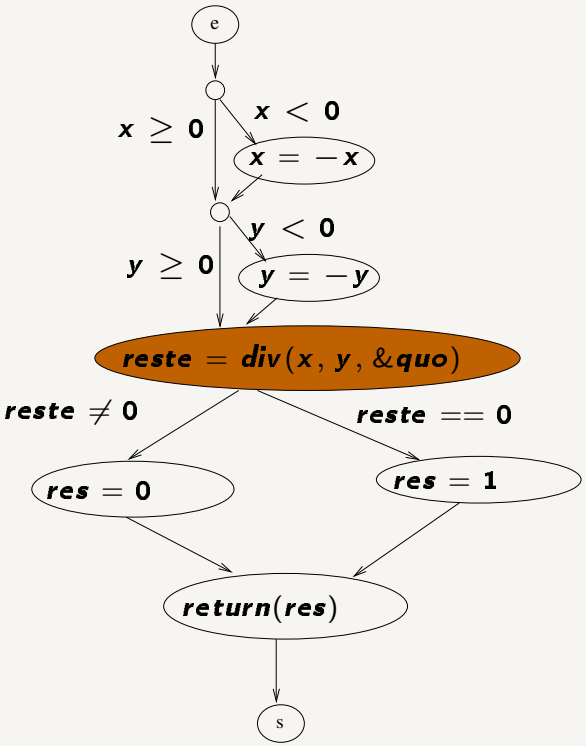
Intêret des contraintes faites sur la spécification

De façon générale

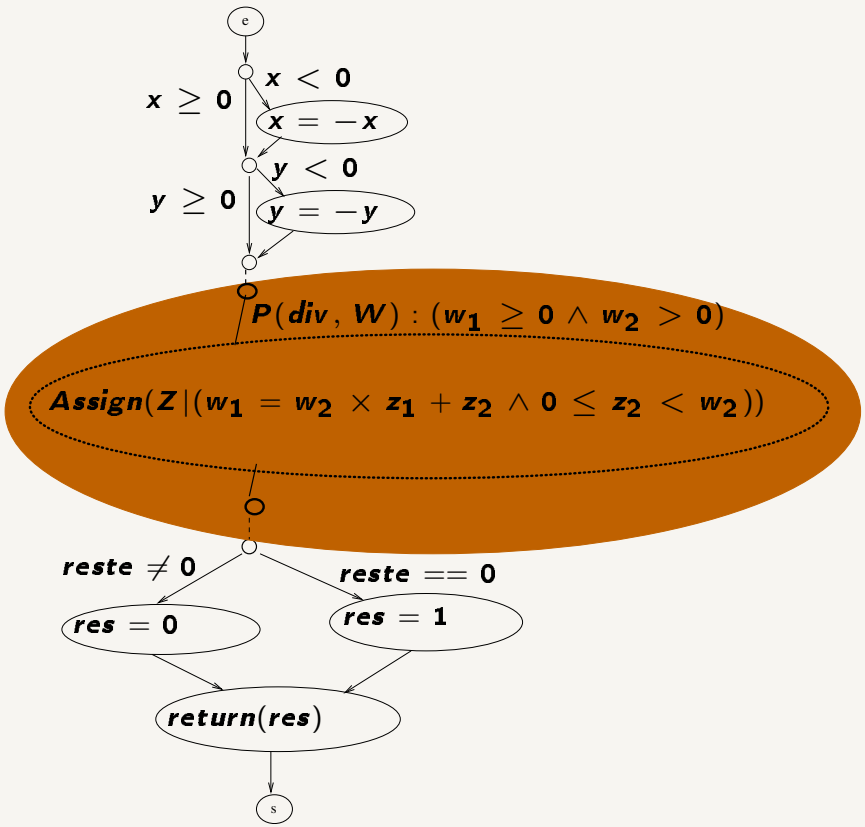


Construction du graphe mixte

CFG de la fonction `est_divisible`



Graphe mixte



Etape suivante

Mise en correspondance des variables fonctionnelles et structurelles

Construction du graphe mixte

Interface

- Fournie par l'utilisateur
- Expertise de l'utilisateur (supposée correcte)
- Indépendante du contexte d'appel

Interface pour la fonction `div`

$INTERFACE : [(w_1, PARAM\ x1), (w_2, PARAM\ x2)], [(z_1, REF\ *q), (z_2, RET)]$

Décomposition

Implantation de la fonction : `int div(int x1,int x2,int *q)`
Décomposition :

- w_1 associée à `x1` (1^{er} paramètre formel)
- w_2 associée à `x2` (2^{nd} paramètre formel)
- z_1 associée à `*q` (3^{eme} paramètre formel avec référence)
- z_2 associée au retour de la fonction

Construction du graphe mixte

Données pour la fonction div et un site d'appel

INTERFACE :
$[(w_1, PARAM\ x1), (w_2, PARAM\ x2)]$,
$[(z_1, REF\ *q), (z_2, RET)]$
Implantation de la fonction :

```
int div(int x1,int x2,int *q)
```

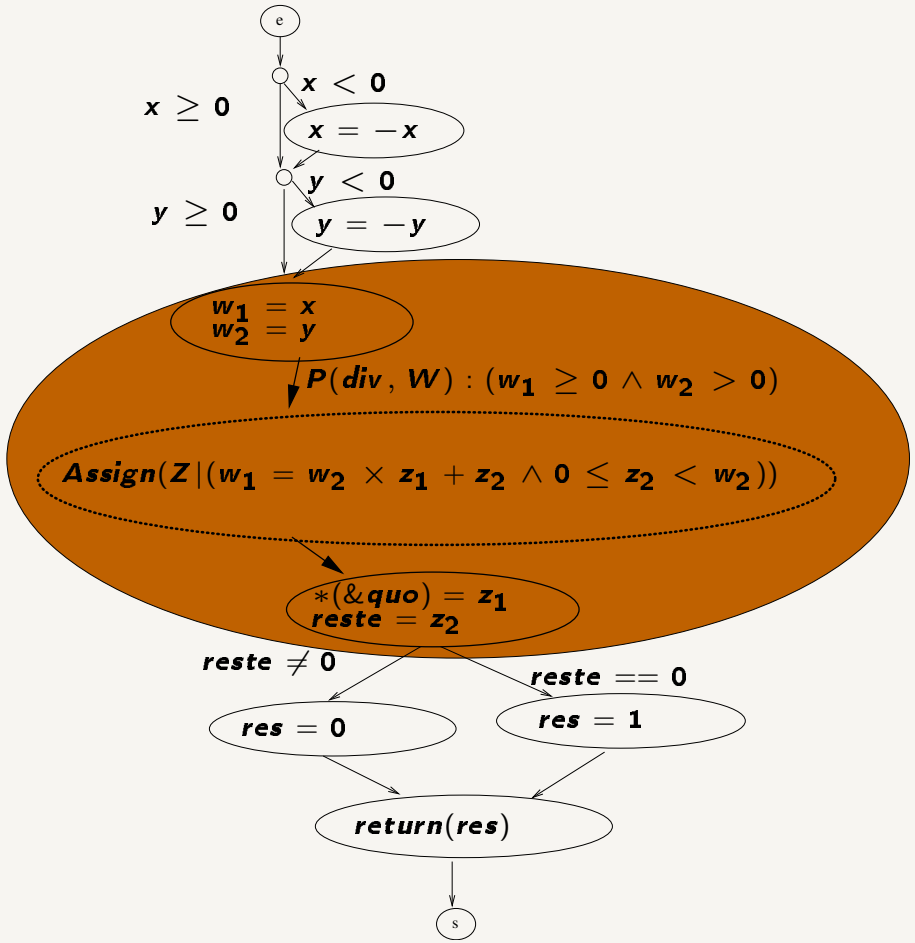
Instruction d'appel :

```
reste =div(x,y,&quo)
```

Besoin

Isoler les instructions d'appel

Graphe mixte de la fonction sous test



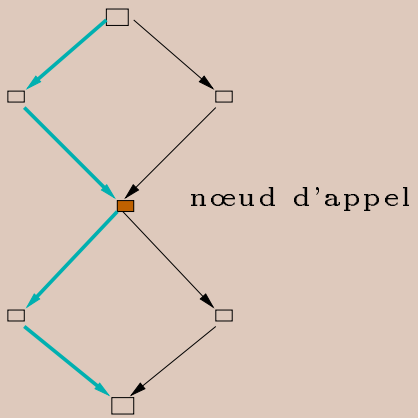
Critères de test associés

Critère TLCM : Tous Les Chemins Mixtes

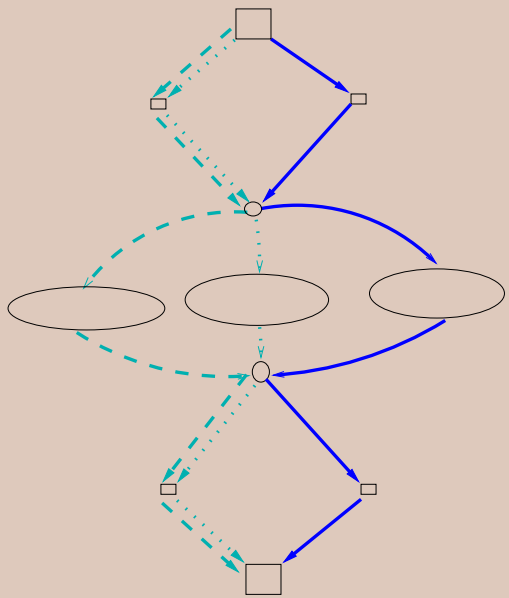
- Critère le plus naturel
- Couverture de tous les chemins du graphe mixte de la fonction sous test (incluant la couverture totale du graphe abstrait).

Redondance dans la couverture du CFG de la fonction sous test

CFG sous test



Critère TLCM



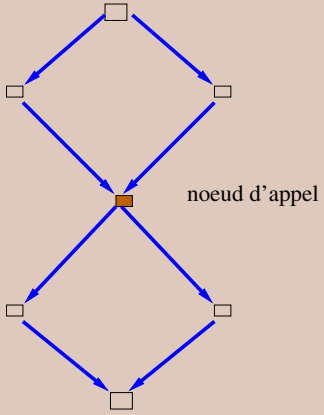
Critères de test associés

Critère QLCS : Que Les Chemins Structurels

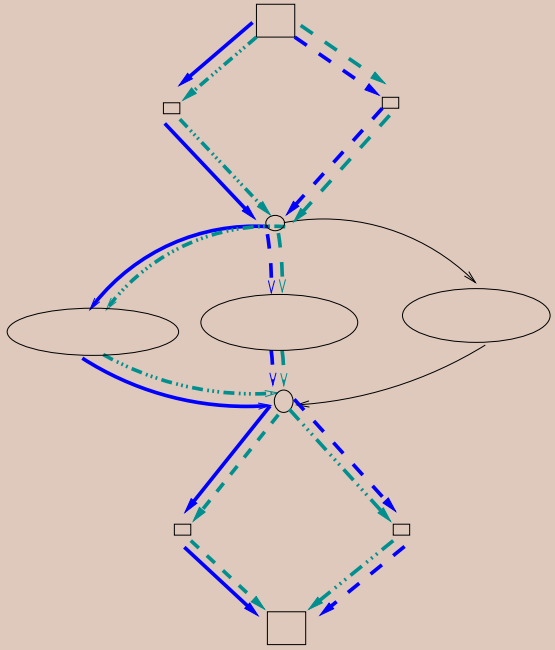
Couverture minimale du graphe mixte correspondant à la couverture de tous les chemins du CFG de la fonction sous test (incluant ou non la couverture totale du graphe abstrait).

Couverture minimale du graphe mixte

CFG sous test



Critère QLCS

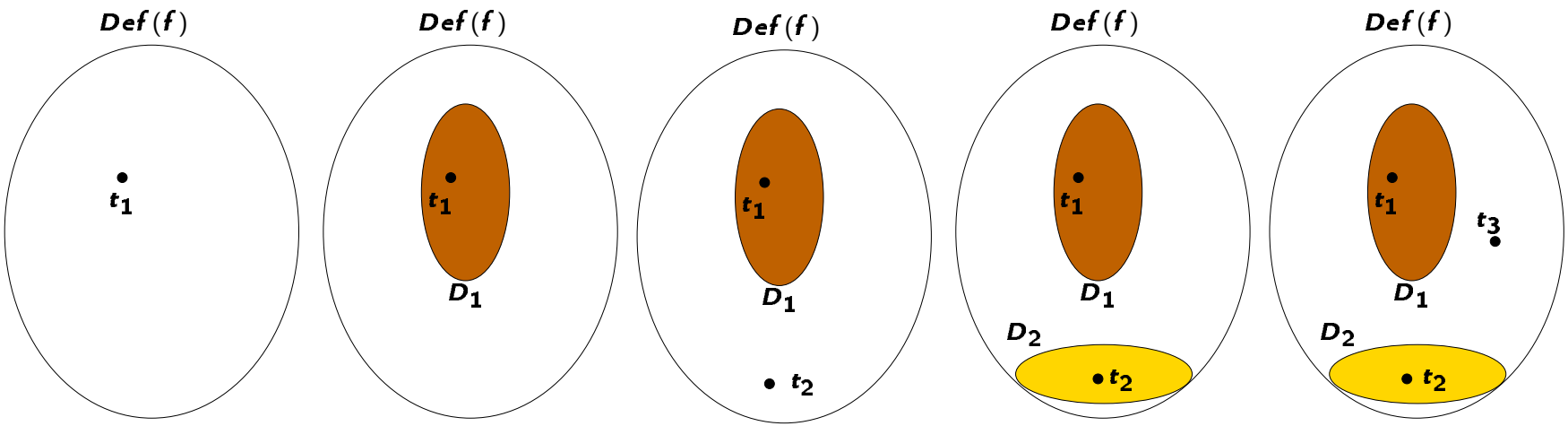


Critère le plus conforme à nos besoins

Mise en œuvre du critère Tous-Les-(k)-Chemins

Stratégie adaptative

1. Exécuter la fonction sous test avec un donnée aléatoire t_1 dans son domaine de définition $Def(f)$
2. Extraire le chemin d'exécution suivi Ch_1
3. Calculer le prédicat de chemin $PC(f, Ch_1, X)$ et le domaine associé D_1
4. Choisir une nouvelle donnée aléatoire t_2 dans le domaine de définition privé du domaine couvert $Def(f) \setminus D_1$
5. Exécuter la fonction sous test avec la donnée d'entrée t_2
6. Itérer les étapes 2 à 5 jusqu'au parcours de la totalité des chemins à couvrir



Comment choisir une donnée de test dans le domaine de sélection ?

Stratégie complète PathCrawler de sélection des données de test

- Réutilisation de la structure du prédicat de chemin courant
- Exploration en profondeur d'abord du graphe
  $\Rightarrow$ exploration du plus long préfixe encore non exploré avec la dernière condition niée

Description du prédicat de chemin

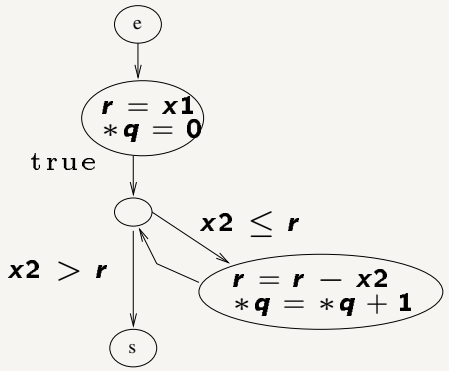
Prédicat $PC_i(f, Ch_i, X)$ exprimé sous la forme d'une conjonction de p conditions ordonnées sur X : $PC_i(f, Ch_i, X) = Cond_1 \wedge ... \wedge Cond_p$

Insastifiabilité

- Insatisfiabilité provient TOUJOURS de la négation de la dernière condition
- Détection simplifiée (heuristiques) mais NP-Complétude (TimeOut)

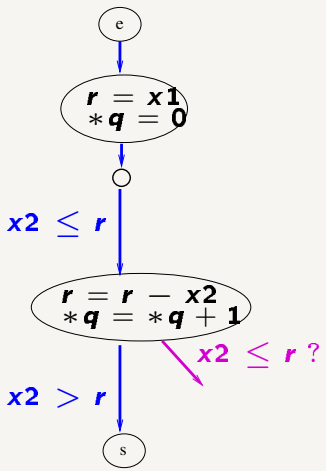
Illustration

CFG de la fonction div



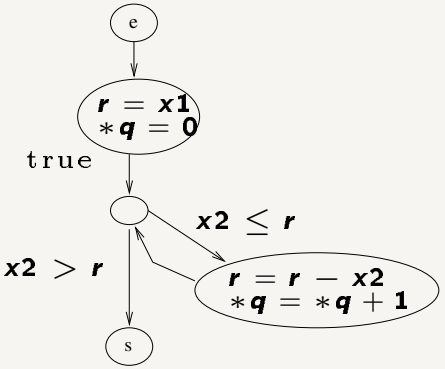
Stratégie de sélection (critère Tous-Les-k-Chemins avec $k = 2$)

[1 itération]

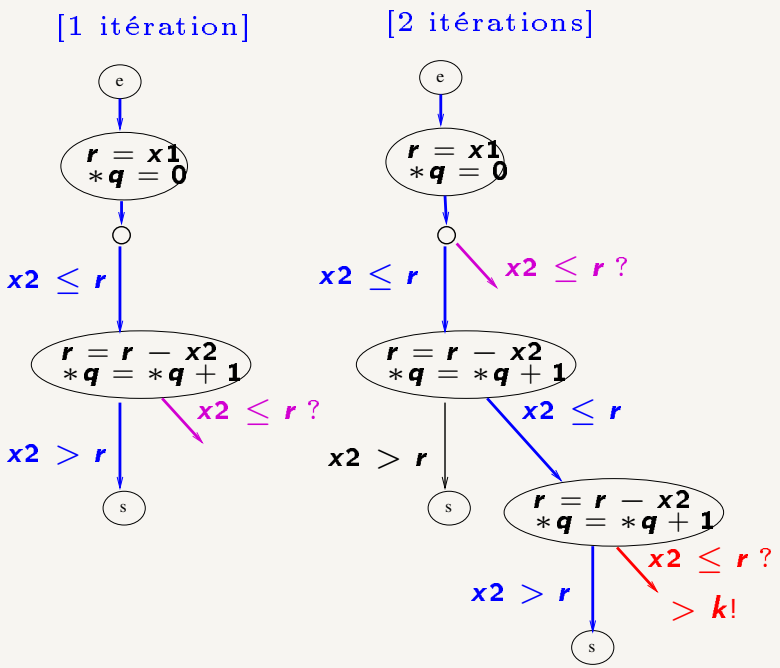


Illustration

CFG de la fonction div

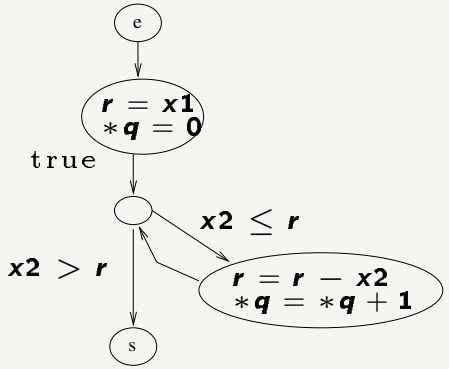


Stratégie de sélection (critère Tous-Les-k-Chemins avec $k = 2$)

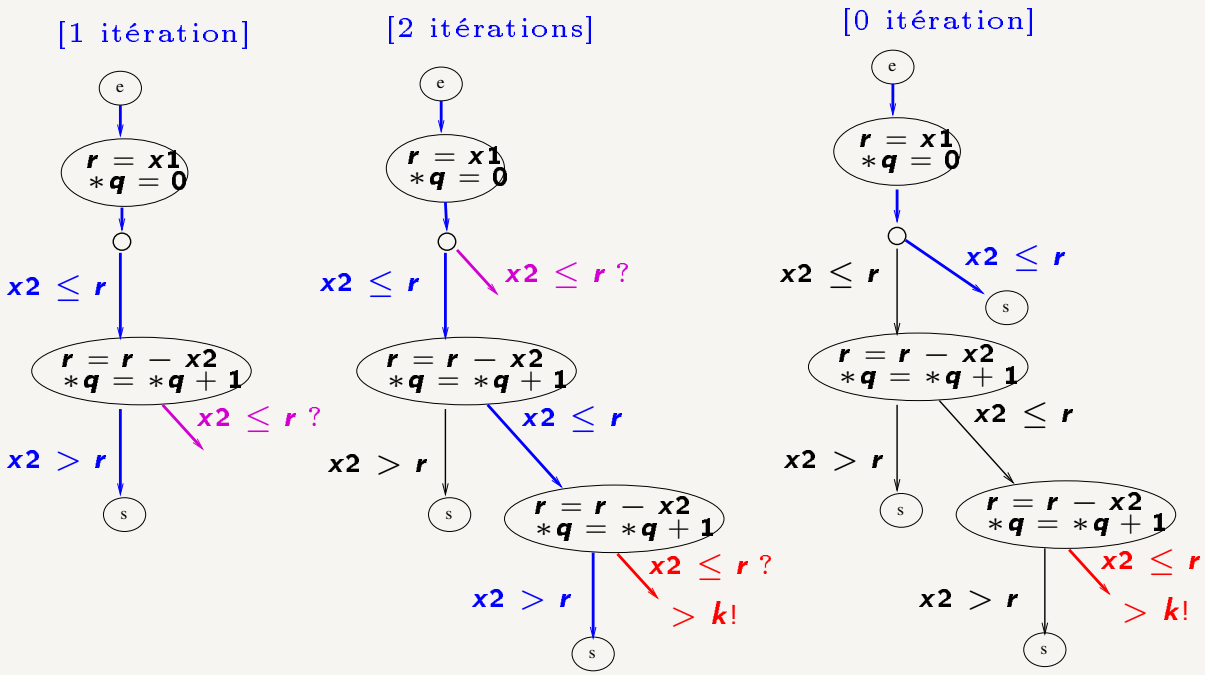


Illustration

CFG de la fonction div



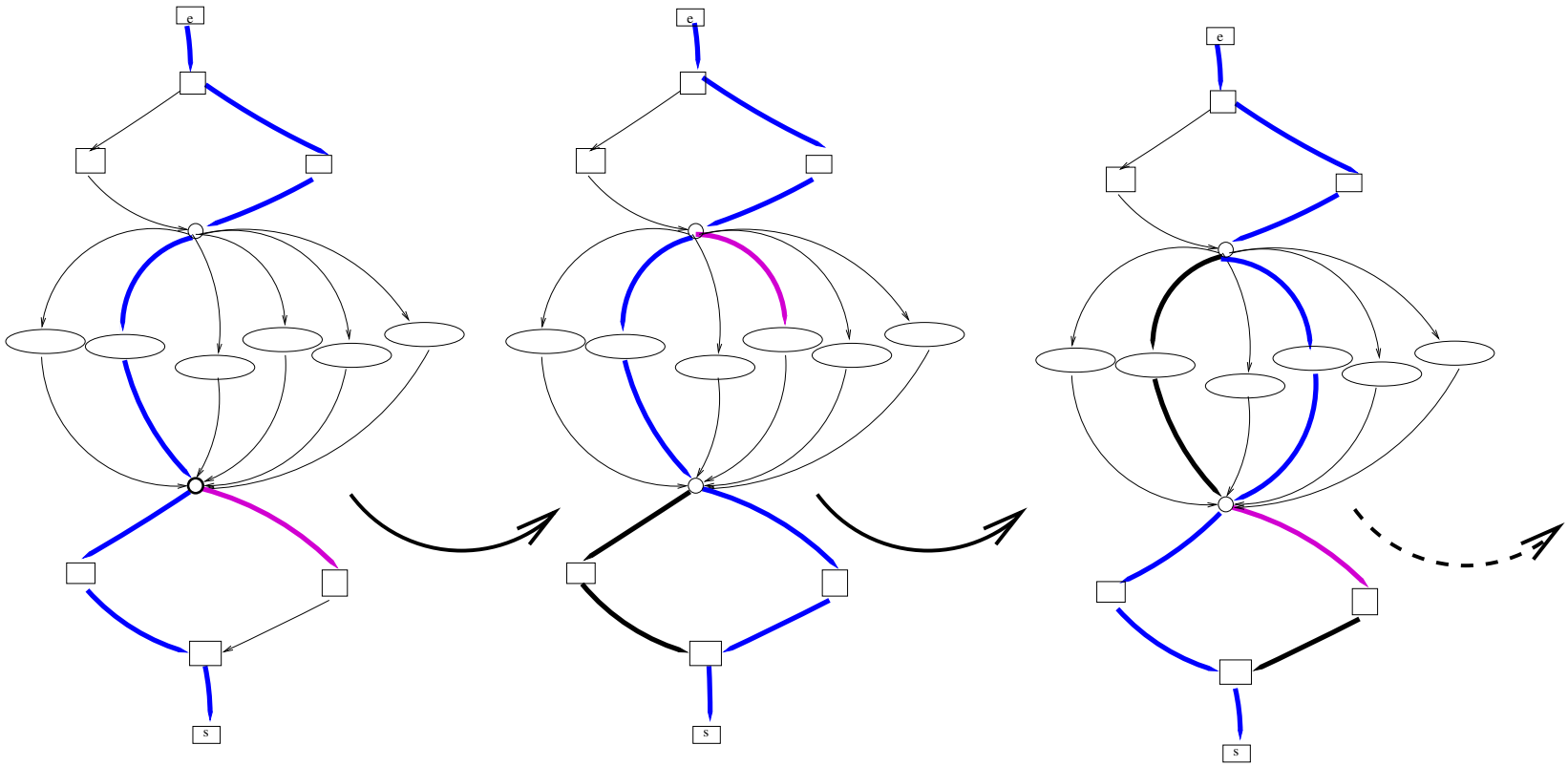
Stratégie de sélection (critère Tous-Les-k-Chemins avec $k = 2$)



Méthode en présence d'appels de fonction

Critère TLCM

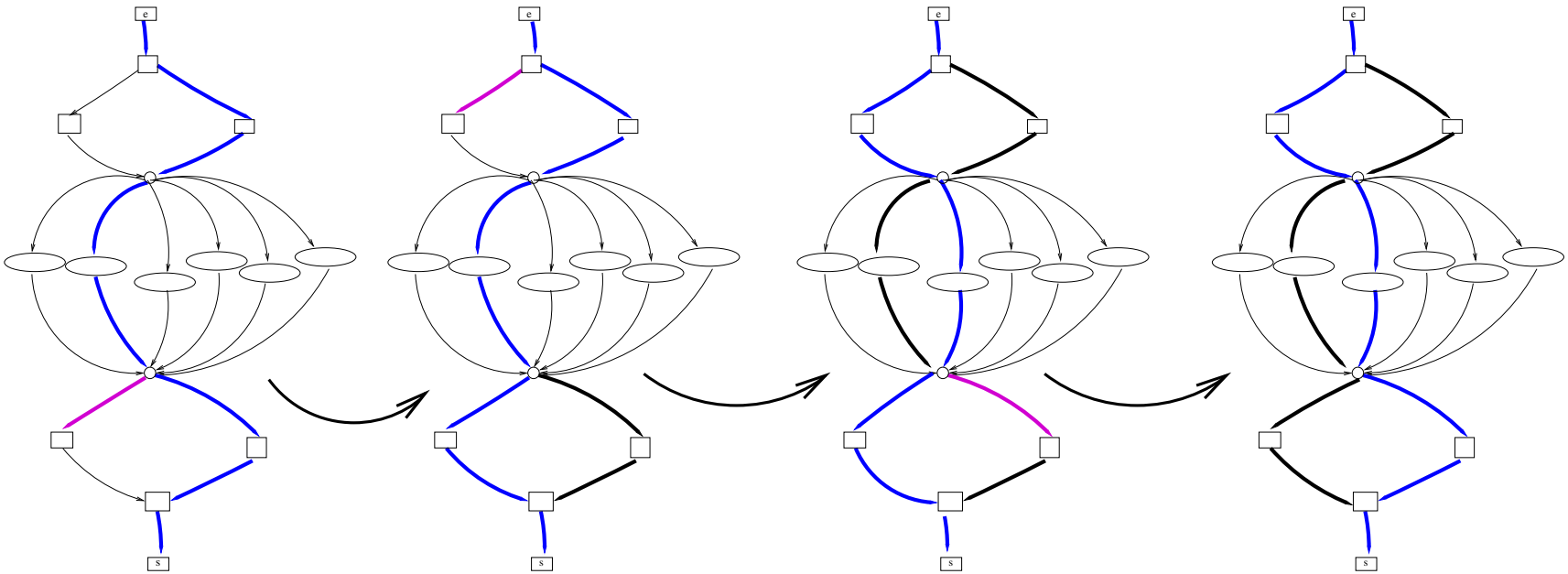
- Immédiat avec la stratégie de sélection de PathCrawler
- TLCM = application du critère Tous-Les-(k)-Chemins sur le graphe mixte de la fonction



Méthode en présence d'appels de fonction

Critère QLCS

Modification de la stratégie de sélection



Modification

- Négation d'une condition interne à la fonction appelée si il existe un chemin "post-appel" non couvert
- Sinon négation d'une condition précédant la fonction appelée

Outil PathCrawler : mise en œuvre de la méthode

Principe

- Génération automatique du test de Tous-Les-(k)-Chemins pour langage C
- Utilisation de la programmation logique avec contraintes (PLC)

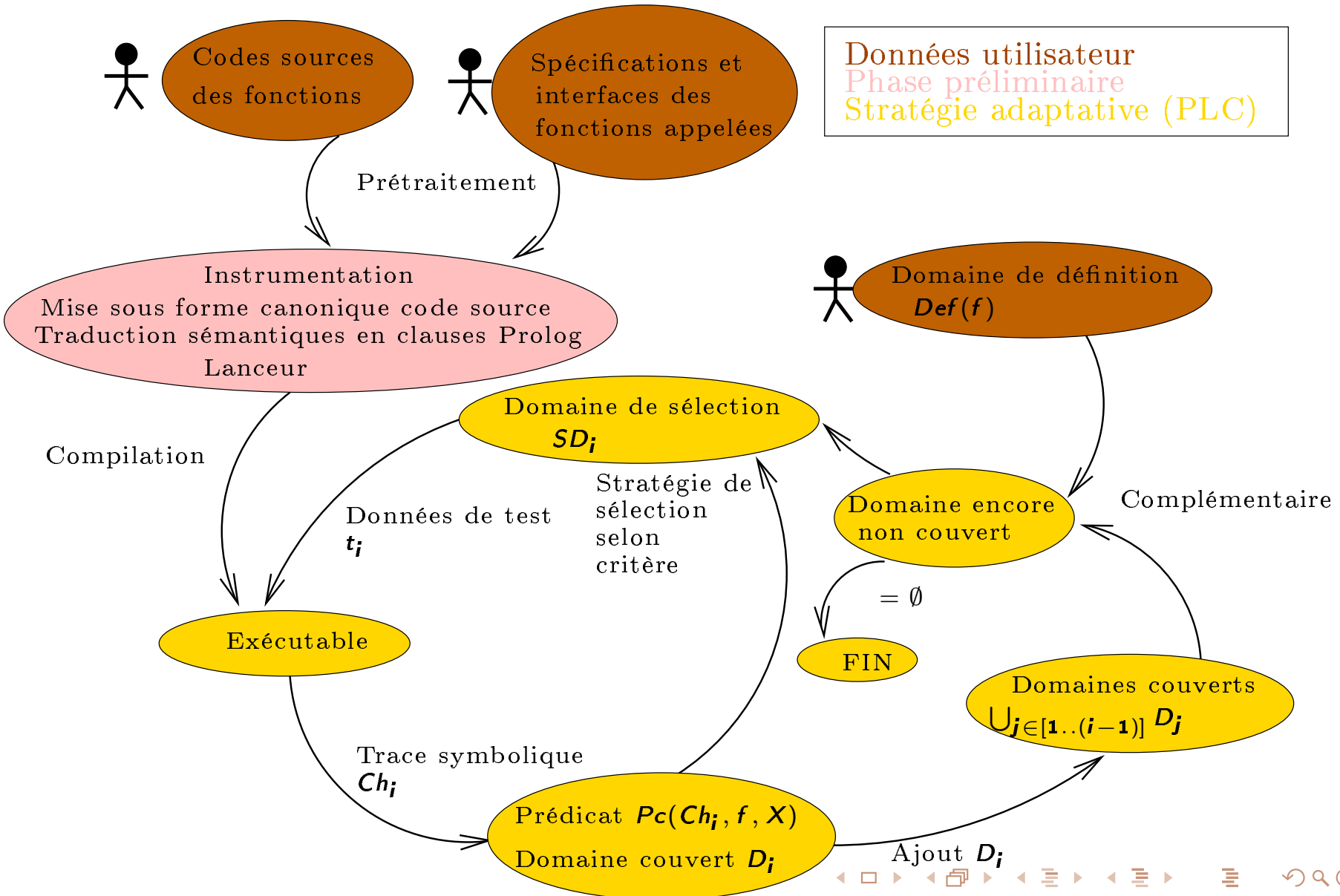
PLC et Test

- Reconnue en test de logiciel : utilisée dans différents outils comme Inka[GBR00], BZ-TT[ABC+02]
- Solveur de contraintes de PathCrawler commun avec l'outil GATEL[MA00]

PathCrawler et PLC

- Calcul du prédicat de chemin
- Caractérisation des domaines de sélection
- Détermination des données de test
- Nécessite la traduction de la sémantique du code C et de la sémantique des spécifications en clauses Prolog

Synthèse de la méthode



Validation 1

Code source fonction sous test

```
int f(int x) {
  if ((x<0))
    x=-x;
  x=m(x);
  if (x<95)
    return(1);
  else
    return(0); }
```

Description de la fonction appelée (fonction macCarthy)

- Fonction appelée utilisée généralement en preuve de programme
- Fonction appelée `m` doublement récursive

Code source fonction appelée

```
int m(int x) {
  if (x>100)
    return (x-10);
  else
    return (m(m(x+11)));}
```

Spécification

```
INTERFACE [(w1,PARAM x)],[(z1, RET)]

FUNCTION m
/*@ requires
@ (w1>=0)
@*/

/*@ ensures
@ ((w1>101) => (z1=w1- 10))
@ ((w1<=101)  => (z1=91))
@*/
```

Résultats exemple macCarthy

Informations

- Fonction appelée : 102 chemins structurels et 2 domaines fonctionnels
- Fonction sous test : 4 chemins (tous avec appel)

	Inlining	Bouchon	TLCM	QLCS
Nbre cas de test	205	2	6	4
% couv. test	100	50	100	100
Cas de test suppl.	201	0	2	0
% Temps[1] (cpu Linux 2GHZ)	100	0.16	0.30	0.23

[1] Génération et exécution des cas de test

Validation 2

Description de la fonction appelée

- Approximation linéaire par morceaux d'une fonction (traitement signal)
- Principe : localisation d'une valeur dans l'intervalle d'un tableau trié

Code fonction sous test

```
int f(int valeur)
{ int i;
  int retour;
  i=g(valeur);
  if ((i==-1)||(i==-2))
    retour=0;
  else
    if ((i>=0)&&(i<n-1))
      retour=1;
    else
      retour=-1;
  return(retour);}
```

Code fonction appelée

```
const int n=25;
const int t[n]={-63,-59,-48,-46,-41,-32,-30,
-27,-21,-15,-9,-3,0,6,11,17,20,28,31,35,
43,44,52,56,60};
int g(int valeur)
{ int r;
  int i;
  if (valeur < t[0])
    r=-2;
  else
    if (valeur>=t[n-1])
      r=-1;
   else
      for(i=0;i<n-1;i++)
        { if ((valeur>=t[i])&&(valeur<t[i+1]))
            r=i;}
  return (r);}
```

Validation 2

Spécification et interface

```
INTERFACE [(w1,GLOB t),(w2,GLOB n),(w3,PARAM valeur)],[(z1,RET)]

FUNCTION g
/*@ requires
@ (true)
@*/

/*@ ensures
@ ((w1[0]>w3) => (z1=-2))
@ ((w1[w2-1]<=w3)  => (z1=-1))
@ ((w1[0]<= w3)&&(w1[w2-1]>w3)  => \exist ind((ind>=0)&&(ind<w2-1)&&
   (w3>=w1[ind])&&(w3<=w1[ind+1]))&& (z1=ind)
@*/
```

Résultats

Informations

- Fonction appelée : 26 chemins structurels et 3 domaines fonctionnels
- Fonction sous test : 3 chemins (tous avec appel)

	Inlining	Bouchon	TLCM	QLCS
Nbre cas de test	26	1	3	3
% couv. test	100	33,33	100	100
Cas de test suppl.	23	0	0	0
Temps (cpu Linux 2GHZ)	100	2.77	5.14	5.14

Comparaisons avec d'autres outils de génération automatique de cas de test unitaires pour langage C

Outil Inka [GBR00]

- Objectif de test et couverture des branches du CFG
- Utilisation de la PLC :
  - Ensemble de la fonction et objectif de test traduits en un système de contraintes
  - Solution du système = donnée de test
- Gestion des appels de fonctions : traitement "inlining"

Outils DART [GKS05] et CUTE [SMA05]

- Critère Tous-Les-Chemins (longueur bornée)
- Stratégie de sélection des données de test en profondeur d'abord (incomplète)
- Gestion des appels de fonctions :
  - CUTE : "inlining" et/ou utilisation des valeurs concrètes en entrée et sortie des fonctions appelées
  - DART (approche SMART [God07]) : fonctions appelées = disjonction de couples (*predicat de chemin*, *sortie symbolique*) et identification d'un chemin de la fonction appelée par ses valeurs concrètes d'appel

Apport : test imbriqué de la fonction appelée

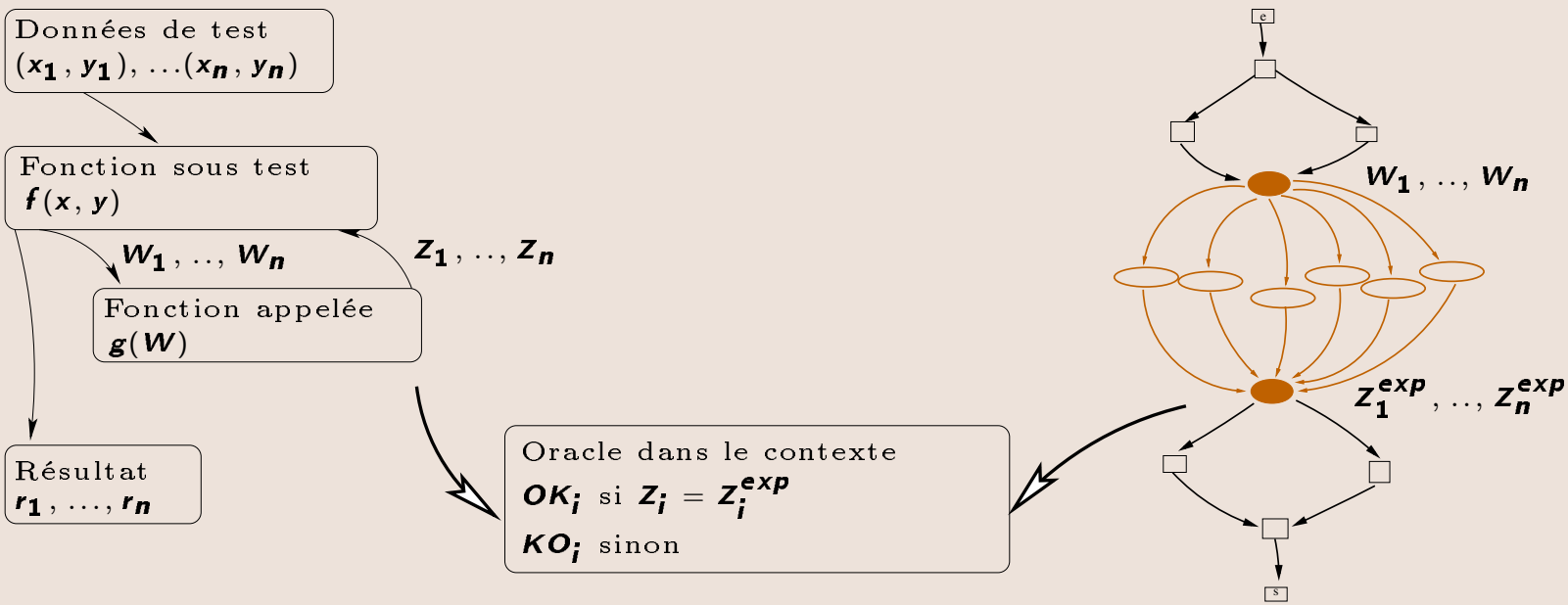
Critère TLCM

Couverture de tous les domaines fonctionnels des fonctions appelées par contexte d'appel

Motivations

- Tester une fonction dans son environnement réel
- Lever une hypothèse : fonction appelée déjà testée

Mise en place d'une technique de test imbriquée



Bilan

Contributions

- Méthode PathCrawler de génération automatique du test Tous-Les-(k)-Chemins étendue à la gestion des appels de fonction
- Implantation d'un prototype : premiers résultats prometteurs
- Nouvelle abstraction mixte des fonctions sous test avec instructions d'appel (aspects structurels et aspects fonctionnels)
- Aide à l'automatisation du test unitaire structurel en présence d'appels : $\Rightarrow$ graphe mixte utilisé comme un CFG classique pour soumission aux différents critères structurels
- Caractérisation de deux nouveaux critères de test :
  - critère TLCM dédié au graphe mixte
  - critère QLCS applicable au CFG déplié de la fonction sous test (traitement inlining)

Perspectives

Extension de l'ensemble du langage C testé

- Validation sur des exemples réalistes
- Arithmétique des pointeurs et types complexes (listes chaînées,...)

Extension du langage de spécification

- Spécifications récursives
- Lever des contraintes sur la spécification (complétude, déterminisme, couples pre/post à domaines exclusifs)

Utilisation conjointe des codes sources et des spécifications pour la génération de tests

- Détermination automatique des variables d'entrée et du domaine de définition
- Mise en place d'un oracle automatique pour la fonction sous test
- Couverture structurelle des domaines fonctionnels (gestion des chemins manquants)

MERCI

Bibliographie

F. Ambert, F. Bouquet, S. Chemin, S. Guenaud, B. Legeard, F. Peureux, N. Vacelet, and M. Utting.
BZ-TT : A tool-set for test generation from Z and B using constraint logic programming.
In *Proc. of Formal Approaches to Testing of Software, FATES 2002 (workshop of CONCUR'02)*, pages 105–120, Brnö, République Tchèque, August 2002. INRIA report.

A. Gotlieb, B. Botella, and M. Rueher.
A CLP framework for computing structural test data.
Lecture Notes in Computer Science, 1861 :399–413, July 2000.

P. Godefroid, N. Klarlund, and K. Sen.
DART : Directed automated random testing.
In *ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI'05)*, pages 213–223, 2005.

P. Godefroid.
Compositional dynamic test generation.
SIGPLAN Not., 42(1) :47–54, 2007.

B. Marre and A. Arnould.
Test sequences generation from Lustre descriptions : GATeL.
In *Proc. ASE 2000*, pages pp 229–237, Grenoble, September 2000. IEEE Computer Society Press.

K. Sen, D. Marinov, and G. Agha.