Fouille de données pour la localisation de fautes dans les programmes

IRISA CNRS UMR 6074

GREYC CNRS UMR 6072

Peggy Cellier – GREYC – Caen

Mireille Ducassé, Sébastien Ferré, Olivier Ridoux – IRISA – Rennes

Contexte

Localisation dynamique de fautes

- **Traces d'exécution des programmes**
  - Contiennent des informations expliquant pourquoi le programme ne renvoie pas le résultat attendu (Défaillances)
  - Sont volumineuses

⇒ **Besoin d'outils d'analyse pour ces traces d'exécution**

Fouille de données (FD)

- **Traitement de grandes quantités d'information**
- **Extraction de régularités dans les données** (règles d'association, ...)
- **Calcul de clusters de données** (analyse de concepts formelle (FCA), ...)
- ...

IDÉE : Utiliser la fouille de données pour analyser les traces d'exécution des programmes

Localisation de fautes avec la FD

Construction du contexte des traces

- **Exécutions du programme contenant une/des fautes**

```
public int Trityp(){
[57] int trityp ;
[58] if ((i == 0) || (j == 0) ||
        (k == 0))
[59]   trityp = 4 ;
[60] else
[61]  {
[62]    trityp = 0 ;
[63]    if ( i == j)
[64]       trityp = trityp + 1 ;
[65]    if ( i == k)
[66]       trityp = trityp + 2 ;
[67]    if ( j == k )
[68]       trityp = trityp + 3 ;
[69]    if (trityp == 0)
[70]    {
[71]      if ((i+j <= k) ||
              (j+k <= i) ||
              (i+k <= j))
[72]          trityp = 4 ;
[73]      else
[74]          trityp = 1 ;
[75]    }
[76]    else
[77]    {
[78]      if (trityp > 3)
[79]          trityp = 3 ;
[80]      else
[81]          if ((trityp == 1)
                    && (i+j > k))
[82]            trityp = 2 ;
[83]          else
// FAUTE (trityp == 2) --> (trityp == 3)
 [84]           if ((trityp == 3)
                    && (i+k > j))
 [85]             trityp = 2 ;
 [86]           else
 [87]             if((trityp == 3)
                     && (j+k > i))
 [88]               trityp  = 2 ;
 [89]             else
 [90]               trityp = 4 ;
 [91]    }
 [92]   }
 [93]   return(trityp) ;}
static public
string conversiontrityp(int i){
[97]  switch (i){
[98]     case 1:
[99]         return "scalen";
[100]    case 2:
[101]        return "isosceles";
[102]    case 3:
[103]        return "equilateral";
[104]    default:
[105]        return "not a ";}}
```

- **Contexte des traces**
  - **Objets :** traces d'exécution
  - **Attributs :** lignes du programme + verdict de l'exécution
  - **Description d'une trace d'exécution :** les lignes exécutées et le verdict de l'exécution
  - **Exemple :** la 1ière exécution exécute les lignes 66, 68, ... est donne un résultat correct : trace e_1

	l_66	l_68	l_81	l_84	l_85	l_87	l_90	$\cdots$	*Succes*	*Echec*
e_1	×	×							×	
e_4	×		×	×		×	×		×	
$\cdots$										
e_{108}		×	×	×	×					×
$\cdots$										
e_{400}									×	

Construction du contexte des défaillances

- **Calcul des règles d'association :** *line* i, $\cdots$, *line* j $\rightarrow$ *Echec*
  - "Quand les lignes i, $\cdots$, j sont exécutées ensemble ***la plupart du temps*** l'exécution produit une défaillance"
  - **Indices statistiques pour mesurer *la plupart du temps***
    - **Support** : fréquence de la règle
    - **Lift** : comment l'exécution de l'ensemble des lignes augmente la probabilité d'avoir une défaillance
- **Contexte des défaillances**
  - **Description des règles par leur prémisse**
  - Règle r_2 :
    line 81, *line* 84, *line* 87, *line* 90, *line* 78, *line* 112, $\cdots$, *line* 93 $\rightarrow$ *Echec*

	l_81	l_84	l_87	l_90	l_105	l_66	l_78	l_112	$\cdots$	l_93
r_2	×	×	×	×	×	×	×	×		×
r_3	×	×	×	×			×	×		×
$\cdots$										
r_{10}							×	×		×

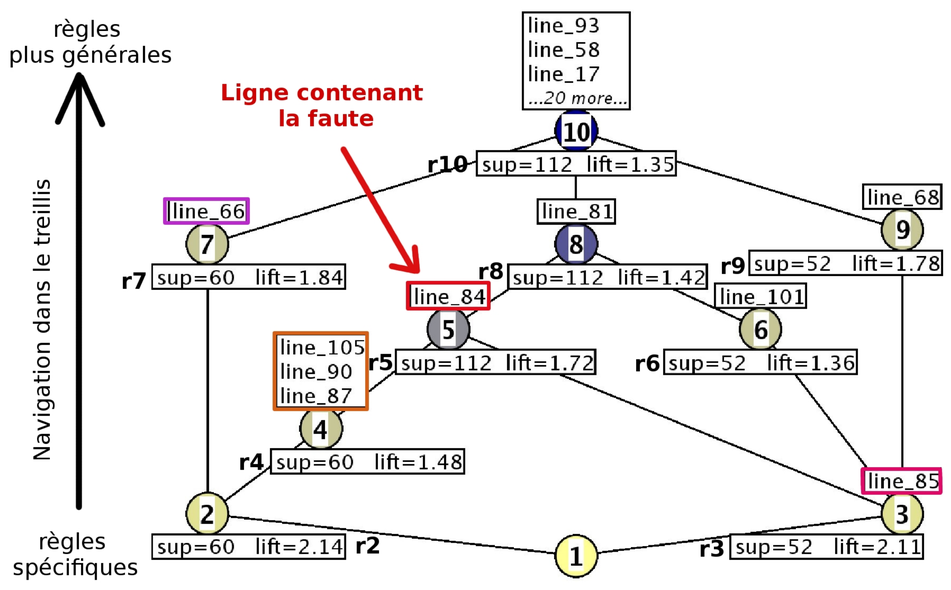
Treillis des défaillances

Beaucoup de règles générées et partiellement ordonnées

⇒ Compréhension des liens entre règles difficile à faire manuellement

IDÉE : Utilisation de la FCA pour construire un treillis

- **Treillis des défaillances à partir du contexte des défaillances**
  - **Concept (cercle) du treillis** : couple (L,R)
    - L : lignes communes aux règles de R
    - R : règles ayant toutes les lignes de L dans leur prémisse
  - Prémisse d'une règle : lignes étiquetant les concepts au dessus du concept étiqueté par la règle
  - Prémisse de r_3 : lignes 85, 84, 68, 101,81, 93, 58, 17, plus les 20 lignes du concept 10.



- **Localisation des fautes**
  - Navigation dans le treillis du bas vers le haut
  - Examen des lignes par un expert pendant la navigation
- **Interprétation de l'exemple**
  - Chemin Concept 2 : `trityp`= 2 et `(i+k>j)`
    → Concept 7 - ligne 66 : initialisation `trityp` à 2
    → Concept 4 - lignes 87, 90, 105 : branche else de la conditionnelle fausse
    ⇒ Résultat : `trityp`= 4 au lieu de 2
  - Chemin Concept 3 : `trityp`= 3 et `(i+k>j)` et `(j+k≤i)`
    ligne 85 : branche then de la conditionnelle fausse
    ⇒ Résultat : `trityp`= 2 au lieu de 4

Avantages de cette approche

- **Pas d'hypothèse sur le type des événements de la trace**
  - Lignes exécutées (exemple ici) mais aussi valeurs des variables, ...
- **Plusieurs exécutions avec des défaillances traitées en une fois**
- **Événements suspects**
  - Apparaissent **souvent** dans des exécution en échec
  - Apparaissent **rarement** dans des exécutions en succès
- **Ordre partiel sur les événements des traces**
  - Prise en compte des dépendances entre les événements des traces

Perspectives

- Utlisation de l'arbre de syntaxe abstraite pour compacter l'information
- Comparaison de suites de tests par comparaison des treillis

[1] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, et Olivier Ridoux, **DeLLIS: a Data Mining Process for Fault Localization**, *SEKE'09*