ASTRÉE

Analyseur Statique de logiciels Temps RÉel Embarqués

Laurent Mauborgne

École Polytechnique — École Normale Supérieure

Mercredi 18 juillet 2005

Grandes lignes

Logiciels critiques

Definition

Logiciels dans lesquels les bugs sont inacceptables

- Trop cher (missions vers Mars, réseau téléphonique AT&T, ...)
- Mettent des vies en danger (missile Patriot, ...)

- Méthodes de production coûteuses
- \+ certification ("preuves" d'absence d'erreurs)

Outils de certification classiques

Inspection manuelle

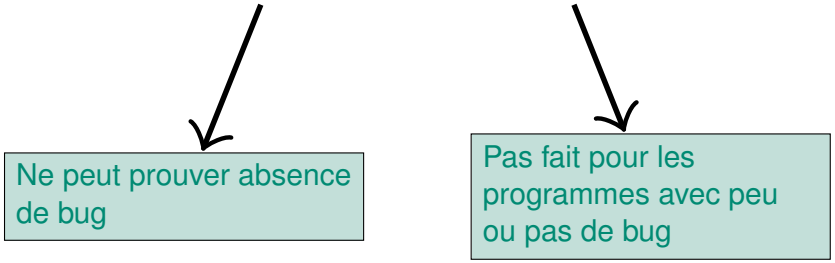
- Très faible confiance
- Ne passe pas à l'échelle

Test

- Pas exhaustif
- Plus le logiciel est gros, moins on couvre
- Confiance minimale trop chère pour logiciels de grande taille

Outils de recherche de bugs

- Automatique (parfois après une phase d'apprentissage)
- Certains sont assez efficaces
- Mais pas exhaustifs et plein de fausses alertes



Nouvelles techniques de certification I
nécessaires pour gros codes critiques

Méthodes déductives

Inspection manuelle assistée par ordinateur

- Modèle d'un programme + preuve
- preuve aidée par ordinateur et vérifiable automatiquement
- Grande confiance dans les résultats (si modèle correct)
- Coût
  - travail d'expert
  - conception d'un modèle par programme
    ⇒ utilisé pour petites parties

Nouvelles techniques de certification II
nécessaires pour gros codes critiques

Analyse statique

Test sur toutes les valeurs possibles

- Modèle d'un langage + analyse automatique mais approchée
- Approximation soulève les questions de
  - correction ?
    ⇒ outils mathématiques
  - précision ?
    si non, fausses alertes
  - en équilibre avec efficacité

Analyse statique de programmes
par interprétation abstraite

But

Découvrir automatiquement les propriétés des programmes

- Souvent indécidables ⇒ approximer
- Interprétation abstraite = cadre mathématique pour
  - approximer souplement les calculs
  - sans introduire d'erreur dans les résultats
  - choisir un équilibre entre précision et efficacité
- Vision unifiée des analyses de programes
  ⇒ factorise les preuves et les améliorations

Conception d'ASTRÉE

- Début en décembre 2001 :
  - Pour gros codes critiques, fausses alertes pas acceptables
  - Airbus demande à l'équipe de Patrick Cousot si peut mieux faire
- Développé par petite équipe :
  - 6 personnes à l'ENS et une à l'X
  - moitié de thésards
- Entre 2002 et 2004, travail sur des version de A340
- Puis, A380 en cours de développement
- Nouveaux contrats avec
  - EDF (centrales nucléaires)
  - Hispano Suiza (moteurs)
  - Astrium (satellites)


Patrick COUSOT


Radhia COUSOT


Jérôme FERET


Laurent MAUBORGNE


Antoine MINÉ


David MONNIAUX


Xavier RIVAL

Erreurs signalées par ASTRÉE

- Interruptions indésirables causées par des exceptions
  - Exceptions sur les opérations de flottants IEEE
    - Opérations invalides ($0/0$, $\sqrt{-1}$)
    - Dépassements (et NaN)
    - Division par zéro
  - Exceptions potentielles causées par des accès mémoires incorrects
    - Accès aux tableaux en dehors des bornes
    - Pointeurs nuls
- Comportements interdits par l'utilisateur
  - Dépassements entier (modulo)
  - Violation d'assertions spécifiées par l'utilisateur

Ce que ASTRÉE peut analyser

- Programmes C synchrones
- Avec
  - Pointeurs (y compris vers fonctions), structures et tableaux
  - Calculs en flottants
  - Tests et boucles
  - Branchements limités (`goto` en avant, `break`, `continue`)
- Sans
  - `union`
  - Allocation mémoire dynamique
  - Appel de fonction récursive
  - Branchement en arrière
  - Effets de bord conflictuels
  - Librairies C

$\Rightarrow$ ASTRÉE peut faire mieux qu'analyseur plus généraliste

Points difficiles qu'ASTRÉE sait gérer

- Erreurs d'arrondi dans les calculs flottants
- Logiciels de grandes tailles (exemple de 400 000 LOC, sans ignorer ni couper aucune partie)
- Très grand nombre de variables globales (exemple avec 20 000, dont 7 000 flottants)
- Flot de contrôle complexe, encodé partiellement avec des variables booléennes non locales

Principe de fonctionnement

Analyse statique par interprétation abstraite

- Couvre tous les comportements possibles
  - Pour capturer **tous** les bugs
  - Obtenu par une exécution abstraite du programme sur toutes ses données possibles à la fois
- Approximation

Sémantique concrète

Comportement des programmes

- domaine concret (état mémoire...)
- fonctions sur ce domaine

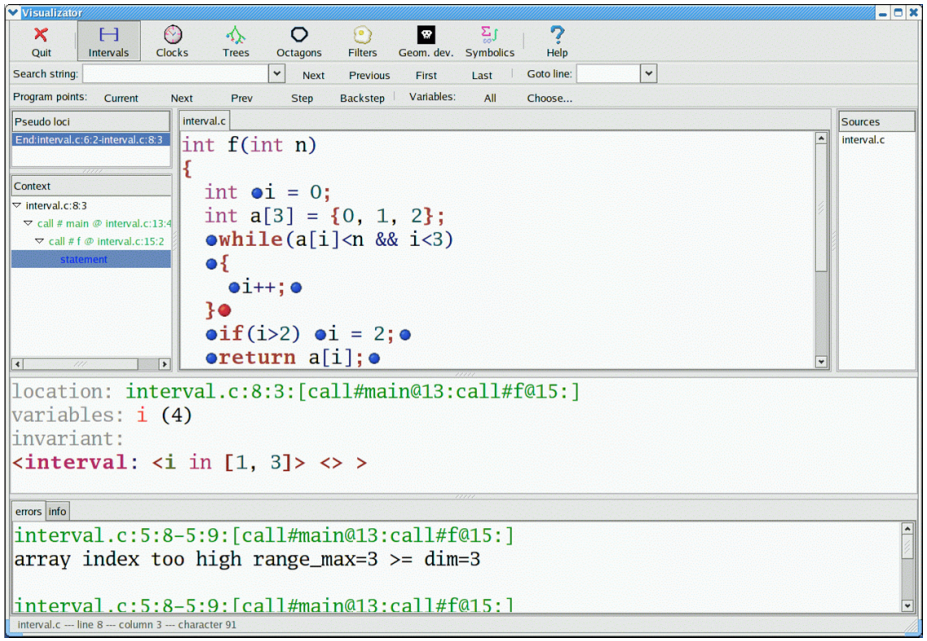
$\xleftarrow{\gamma}$

Sémantique abstraite

Simule le comportement réel

- domaine abstrait
- fonctions abstraites
- élargissement (approximation grossière de l'union)

Visualisation dans ASTREE



Exemple

Analyse avec des intervalles

Exemple

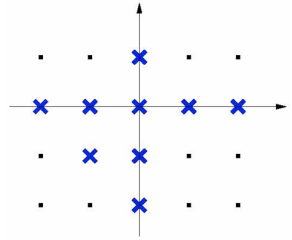
```
int f(int n) {
  int i=0;
  int a[3] = {0,1,2};
  while(a[i]<n && i<3) {
    i++;
  }
  if(i>2) i=2;
  return a[i];
}
```

- $n \in [-2147483648, 2147483647]$
- $i \in \{0\}$
- Après itération, en début de boucle, $i \in [0, 3]$
- Phase de signalement : alerte sur l'accès au tableau

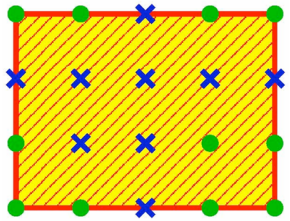
Des approximations souples

- Domaine abstrait
  - seules les propiétés du domaine sont représentables
  - approcher les ensembles d'états concrets
  - objectif : garder les propriétés intéressantes et représentables
  - pas d'obligation de clôture algébrique
- Fonctions abstraites
  - traduire les instructions élémentaires du programme
  - objectif : précision et rapidité
- Élargissements
  - objectif : terminer en temps raisonable
- Décomposition en plusieurs domaines abstraits
  - simplification de l'implémentation
  - réutilisabilité
  - mais approximations lors des transferts d'informations

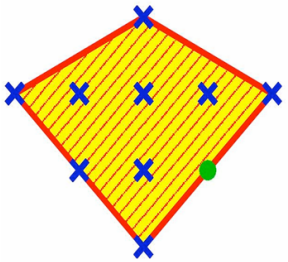
Domaines abstraits



Ensemble d'états mémoire concrets



Représentation par intervalles

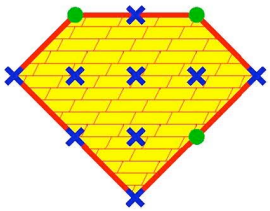


Représentation par polyèdre convexe

Analyse précise $\Rightarrow$ garder des relations entre variables ($x < 2y$, $xy > z$,...)

Octogones

- Analyse relationnelle assez chère (polyèdres convexes : exponentiel)
- **Octogones** : une des représentations relationnelles les plus économes pour les tuples de nombres (taille $O(n^2)$, opérations $O(n^3)$)
- Malgré tout, besoin d'analyse hétérogène (packs de variables)
- Relations sont : $x \pm y$ dans intervalle



Exemple avec les octogones

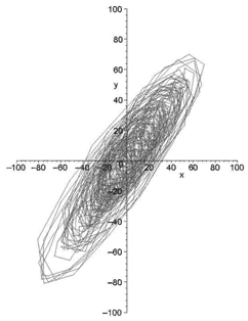
Exemple

```
int f(int n) {
  int x = n;
  int y = n-1;
  while(x > 2) {
    x = x-1;
    y = y-1;
  }
  return x-y;
}
```

- Alerte si pas d'hypothèse sur les entrées
- Octogone : `x-y=` 1
- Juste intervalles : `x<` 3
- Avec octogones, `f` renvoie 1
- Sans, alerte

Filtres

- Dans les programmes command/control, utilisation intensive des filtres numériques
- Invariants des filtres trop complexes pour les représentations symboliques linéaires
- ASTRÉE prend en compte très précisément les erreurs d'arrondi des calculs flottants
- Domaine de filtres générique implémenté dans ASTRÉE, permettant ajout facile de nouveaux types de filtres



Exemple avec les filtres

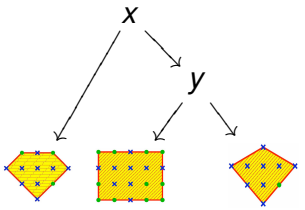
```
void f(double v) {
  double E1=0, E2=0;
  double S0=0, S1=0, S2=0;
  while(randomBool()) {
    E0 = randomDoubleIn(-1, 1);
    if(randomBool()) {
      S0 = E0;
      S1 = E0;
      E1 = E0;
    }
    S0 = 1.5*S1 - 0.7*S2
     + 0.5*E0  - 0.7*E1 + 0.4*E2;
    E1 = E0;
    S2 = S1;
    S1 = S0;
  }
```

- Pas d'approximation par contraintes linéaires
- Contraintes ellipsoïdes
- Approximation de l'expansion formelle des sorties en tant que somme des entrées

Arbres de décision

- Beaucoup de booléens utilisés pour contrôler le flot

  $\Rightarrow$ besoin de relier les variables booléennes et numériques
- Arbres de décision :
  - nœuds étiquetés par variables booléennes (décisions),
  - feuilles étiquetées par représentations symboliques des valeurs numériques

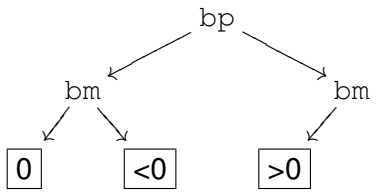


POLYTECHNIQUE

Exemple d'arbre de décision

```
double f(int x) {
  double y;
  int bp, bm;
  bp = (x>0);
  bm = (x<0);
  if(!bp && !bm) {
    y = 0;
  } else {
    y = 1.0/x;
  }
  return y;
}
```

- En suivant l'arbre de décision, on obtient `x==0` après le test
- (fausse) alerte sans arbre de décision
- Paramètre : ♯ de variables booléennes par pack



Distinction des traces

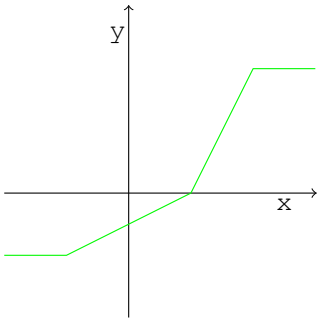
- Trace = séquence d'états mémoire d'une exécution particulière
- Des approximations viennent de la fusion des traces
  - Après les deux branches d'un test
  - Dans les boucles
  - Car les valeurs sont fusionnées dans les représentations symboliques
- Partitionnement de traces pour regagner de la précision

POLYTECHNIQUE

Interpolation Linéaire

Example

```
float[] tc = { 0;  0.5;  2; 0};
float[] tx = { 0;   -1;  1; 2};
float[] ty = {-1; -0.5; -1; 2};
int i = 0;
•while (i<3 && x>tx[i+1]) i++;•
•y = tc[i] * (x - tx[i]) + ty[i];•
```



- **Intervalles :** $i \in \{0\}$ et $x \in [-2,2]$ $i \in \{1\}$ et $x \in [-1,2]$ $i \in [0,1]$ et $x \in [-2,2]$ $i \in [1,2]$ et $x \in [-1,2]$ Point fixe : $i \in [0,3]$ et $x \in [-2,2]$ Sortie de `while` : $i \in [0,2]$ et $x \in [-2,2]$ $i \in [0,2]$ et $x \in [-2,2]$ et $y \in [-4,4]$
- **Distinction de traces :** $i \in \{0\}$ et $x \in [-2,2]$ Boucle 1 : $i \in \{1\}$ et $x \in [-1,2]$

Performances d'ASTRÉE

- Pas fonction monotone des LOCs !
- Assez efficace pour utilisation en phase de développement :

Programmes	test 1	test 2	test 3	test 4
Taille (LOCs)	70 000	65 000	215 000	380 000
Itérations	**43**	**32**	**59**	**62**
Analyse (minutes)	**70**	**28**	**330**	**680**
Pics mémoire (Mb)	**550**	**390**	**1 300**	**2 200**
Alertes	**0**	**2**	**1**	**0**

- Entièrement automatique sur cette famille de logiciels
- ASTRÉE peut aussi tourner en parallèle

Industrialisation

- ASTRÉE bien adapté aux logiciels critiques airbus
  - Développement de domaines spécifiques (filtres,...)
  - Prise en charge de leurs sous-ensemble de C
  - Réglage des quelques 150 paramètres (par famille de logiciels)
    - Packs pour les octogones et arbres de décision
    - Points de distinction de traces
    - Stratégie d'itération pour point fixe précis
- *A contrario*, l'extension à d'autres utilisateurs va nécessiter
  - Nouveau modèle mémoire (`union`)
  - Nouveaux domaines abstraits
  - Nouveaux réglages
  - Si alertes, recherche de bug ou d'imprécisions
- Requiert connaissances de base en interprétation abstraite
- Recherche active motivée par les applications

Conception de domaines abstraits

Préoccupations

- Compacité des données
- Expressivité
- Efficacité des algorithmes

⇒ réutilisation de structures efficaces de la littérature

Algorithmique à explorer

- Possibilité d'approximer certains algorithmes
- Pas de contrainte de fermeture algébrique

Challenges

Challenges à court terme

- Extension du modèle mémoire en cours
- Analyse précise des structures de données dynamiques
- Rapports détaillés sur les alertes (explication de l'origine)
- Migration de l'analyse au niveau de l'assembleur

Challenges à moyen terme

- Analyse des spécifications
- Preuve de propriétés complexes de l'utilisateur
- Analyse de programmes asynchrones
- Preuve formelle de correction d'ASTRÉE

Conclusion

- ASTRÉE montre qu'il est possible de faire
  - Analyse automatique
  - Sur codes de taille industrielle
  - Efficace

  $\Rightarrow$ plus d'excuse pour des fautes de logiciels
- S'appuie sur théorie informatique formelle, l'interprétation abstraite

Industrialisation très prochaine

- Besoin de chercheurs
- Besoin d'utilisateurs formés

  $\Rightarrow$ venez au cours d'interprétation abstraite en majeure !