L'Analyseur Statique ASTRÉE

Patrick Cousot

École normale supérieure, Paris

cousot@ens.fr www.di.ens.fr/~cousot

Grand Colloque TIC 2006, Session RNTL « Systèmes embarqués », Centre de congrès, Lyon, 15 novembre 2006

Plan

Résumé

Nous rappelons les objectifs du projet RNTL ASTRÉE (*Analyse Statique de Logiciel Temps Réel Embarqué*, déc. 2002—mars 2006) sur la vérification complètement automatique d'absence d'erreurs à l'exécution dans les codes embarqués de contrôle/commande, puis expliquons les bases formelles de l'interprétation abstraite utilisées pour concevoir l'analyseur statique ASTRÉE (www.astree.ens.fr), en particulier l'adaptation à un domaine d'application spécifique, les résultats obtenus par l'analyseur statique ASTRÉE sur les logiciels de commande de vol électrique de l'A340 et de l'A380, les perpectives industrielles et les développements scientifiques à moyen terme, en particulier en ce qui concerne l'analyse statique de programmes asynchrones (projet RNTL THÉSÉE).

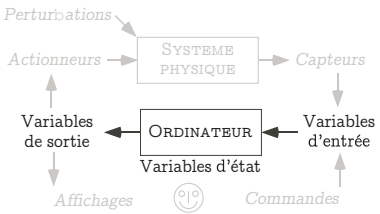
Une courte démonstration de l'analyseur statique ASTRÉE est prévue pendant la pause.

Objectifs du projet RNTL « ASTRÉE »

Analyse statique de programmes synchrones

– Programmes C embarqués de contrôle/commande temps-réel synchrone :

```
déclarer et initialiser les variables
d'état;
loop forever
  lire les variables d'entrée volatiles,
  calculer les variables de sortie et
  d'état,
  écrire les variables de sortie;
  attendre le prochain tick d'horloge
end loop
```



Introduction informelle à l'interprétation abstraite

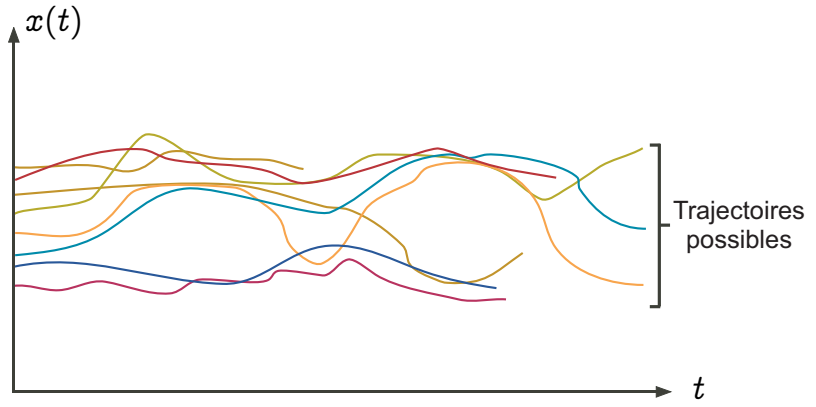
Objectif d'ASTRÉE

– Démontrer automatiquement l'absence d'erreurs à l'exécution :

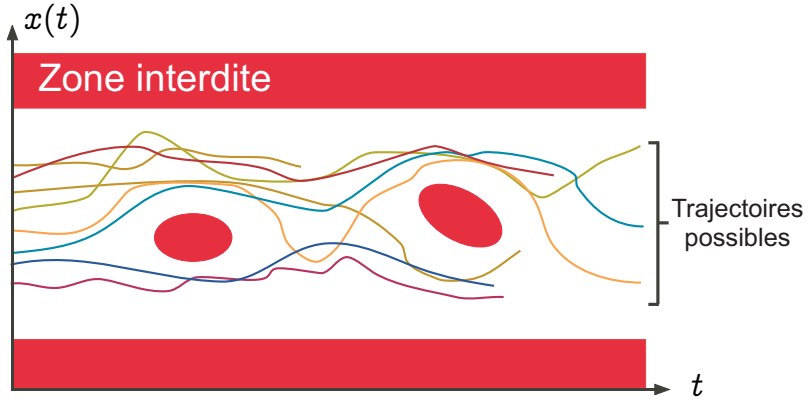
– Exigences :
  - correction (pas d'erreur oubliée)
  - passage à l'échelle (100 000 à 1000 000 LOCs)
  - efficacité (analyse faisable sur une station de travail)
  - précision (peu de fausses alarmes)

– Pas d'alarme → certification complète

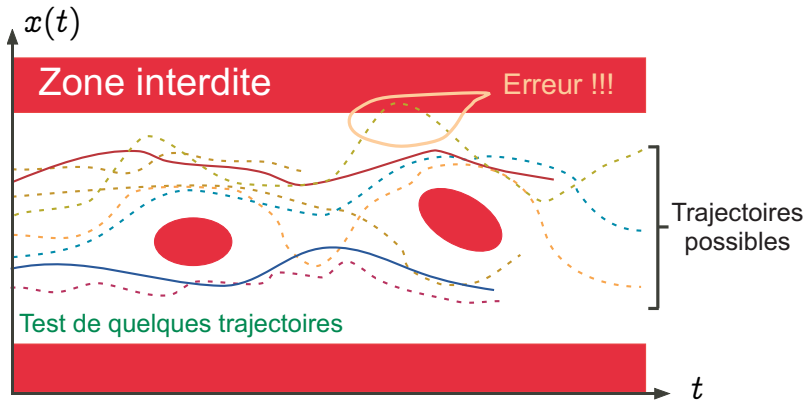
Sémantique opérationnelle



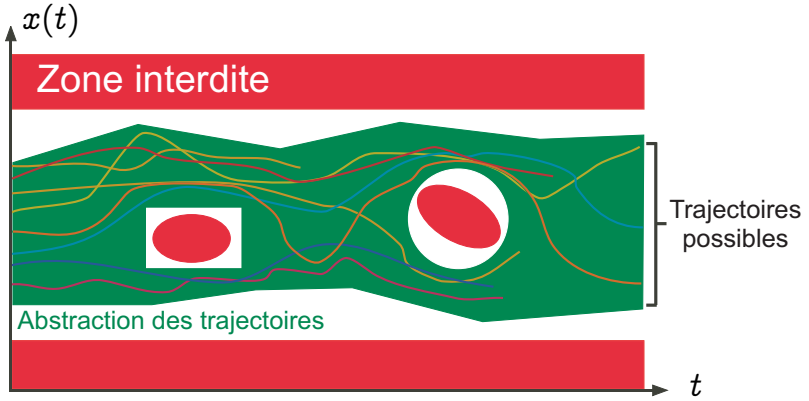
Propriétés de sûreté



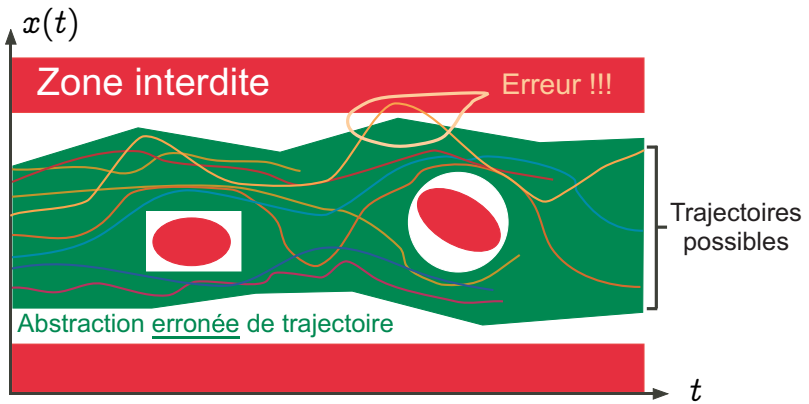
Le test n'est pas sûr



L'interprétation abstraite est sûre

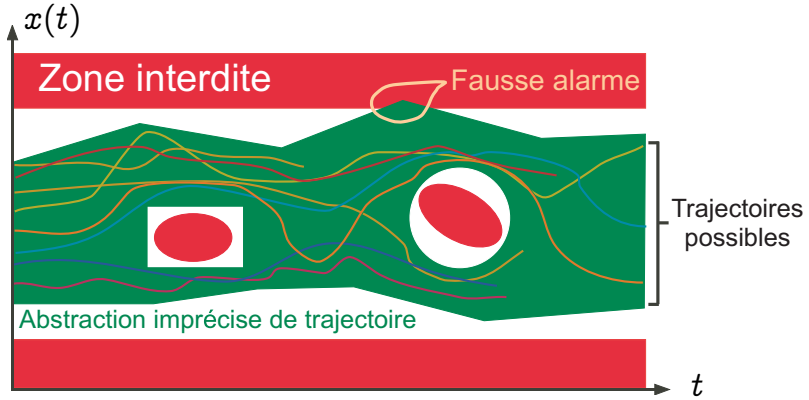


Exigence de sûreté : abstraction erronée [1]

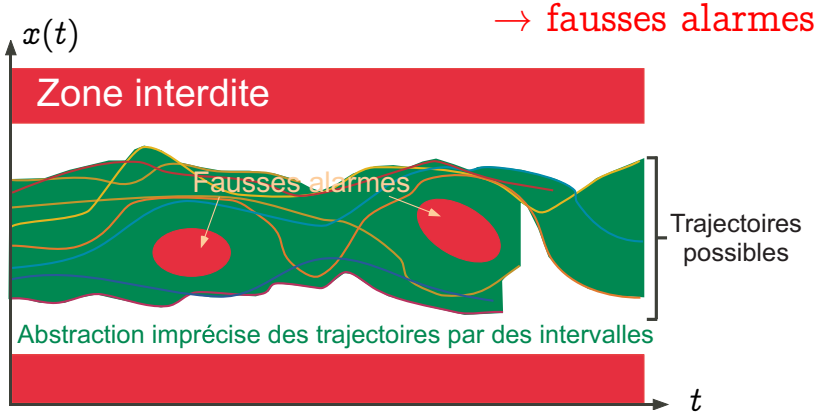


[1] Cette situation est toujours exclue en analyse statique par interprétation abstraite.

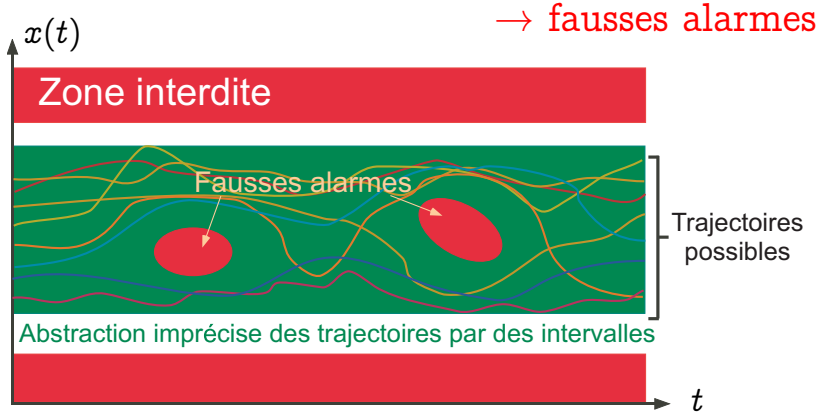
L'imprécision entraîne des fausses alarmes



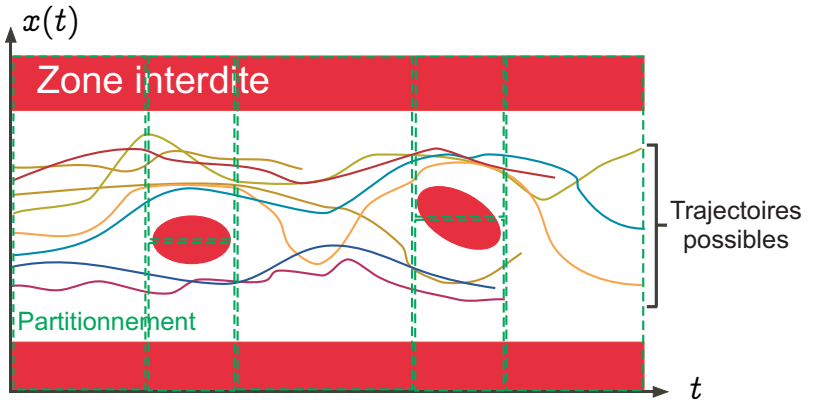
Abstraction locale par intervalles → fausses alarmes



Abstraction globale par intervalles → fausses alarmes

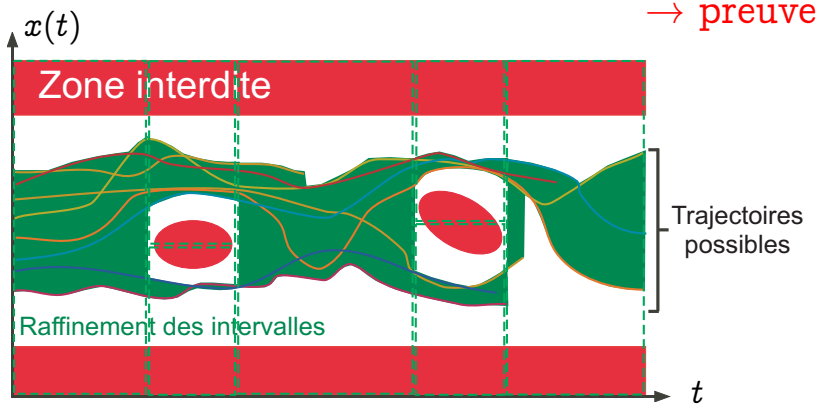


Raffinement par partitionnement



Intervalles avec partitionnement

→ preuve

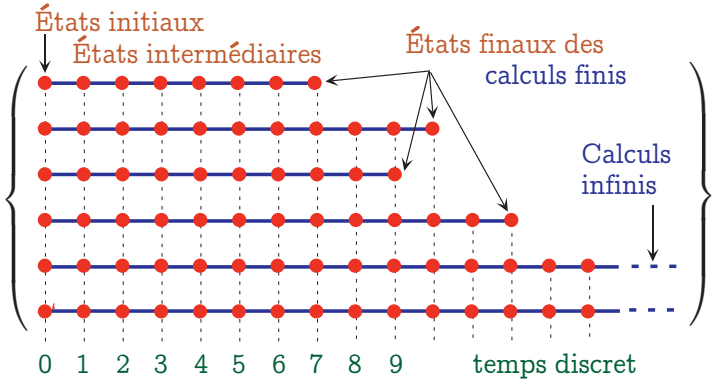


Conception d'un analyseur statique par interprétation abstraite

Sémantique du language

1. Définir la sémantique du language $\mathcal{S} \in \mathcal{L} \mapsto \mathcal{D}$ et les propriétés concrètes $\wp(\mathcal{D})$;

Exemple : sémantique de traces



États $\Sigma = \{\bullet, \ldots, \bullet \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \bullet \ldots\}$

Sémantique de traces, formellement

– Traces d'un système de transitions $\langle \Sigma, \tau \rangle$:

finies $\Sigma^+ \stackrel{\text{def}}{=} \bigcup_{n>0} [0, n[\mapsto \Sigma$, infinies $\Sigma^\omega \stackrel{\text{def}}{=} \mathbb{N} \mapsto \Sigma$

– $\mathcal{D} = \wp(\Sigma^+ \cup \Sigma^\omega)$ domaine sémantique

– $\mathcal{S}[\![\langle \Sigma, \tau \rangle]\!] = \mathsf{lfp}^{\sqsubseteq}\ F \in \mathcal{D}$ sémantique de traces

– $F(X) = \{s \in \Sigma^+ \mid s \in \Sigma \wedge \forall s' \in \Sigma : \langle s, s' \rangle \notin \tau\}$

$\cup \{ss'\sigma \mid \langle s, s' \rangle \in \tau \wedge s'\sigma \in X\}$ opérateur

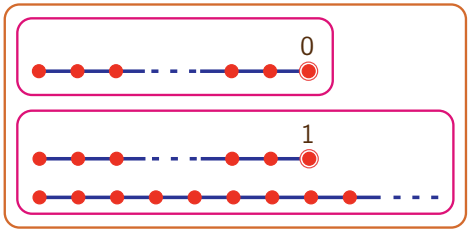
– $X \sqsubseteq Y \stackrel{\text{def}}{=} (X \cap \Sigma^+) \subseteq (Y \cap \Sigma^+) \wedge (X \cap \Sigma^\omega) \supseteq (Y \cap \Sigma^\omega)$

ordre de calcul

Propriétés à démontrer

2. Définir la propriété à démontrer $\mathcal{Q}[\![P]\!] \in \wp(\mathcal{D})$ des programmes P : $\mathcal{S}[\![P]\!] \in \mathcal{Q}[\![P]\!]$

Exemple de propriété de programme



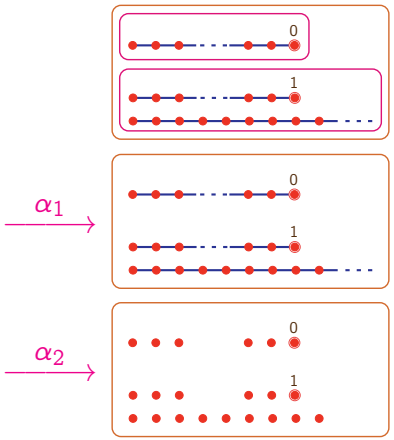
– Implantations correctes : `print 0`, `print 1`, `[print 1|loop]`, ...

– Implantations incorrectes : `[print 0|print 1]`

Abstraction

3. Choisir l'abstraction $\langle \wp(\mathcal{D}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$

Exemple d'abstraction : états accessibles

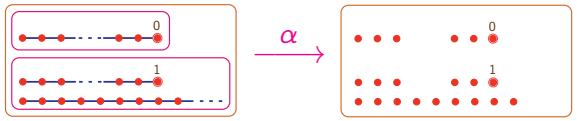


Sémantique abstraite

4. La théorie de l'interprétation abstraite permet de construire formellement une sémantique abstraite $\mathcal{S}^\sharp[\![P]\!] \sqsupseteq \alpha(\{\mathcal{S}[\![P]\!]\})$

Exemple : états accessibles

– Système de transition : $\langle \Sigma, \tau \rangle$
– États initiaux : $\mathcal{I} \subseteq \Sigma$
– Abstraction :



– États accessibles :

$$\alpha(\{\mathcal{S}[\![\langle \Sigma, \tau \rangle]\!]\}) = \alpha(\mathsf{lfp}^{\sqsubseteq} F) = \mathsf{lfp}^{\subseteq} F^\sharp,$$
$$F^\sharp(X) = \mathcal{I} \cup \{s' \mid \exists s \in X : \langle s, s' \rangle \in \tau\}$$

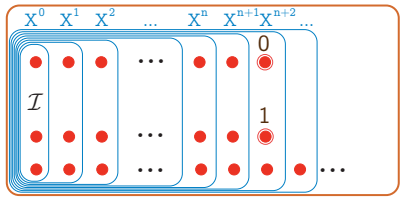
Analyseur statique

5. L'algorithme d'analyse statique est le calcul de la sémantique abstraite (donc correct par construction)

Exemple : états accessibles

– Système de transition : $\langle \Sigma, \tau \rangle$
– États initiaux : $\mathcal{I} \subseteq \Sigma$
– États accessibles : $\mathsf{lfp}^{\subseteq} F^\sharp = \bigcup_{n \geq 0} F^{\sharp n}(\emptyset)$, où

$$F^\sharp(X) = \mathcal{I} \cup \{s' \mid \exists s \in X : \langle s, s' \rangle \in \tau\}$$



itération infinie
$\Rightarrow$ accélération
de la convergence

Résultats de l'analyse statique

6. Le résultat du calcul est
   – $\mathcal{S}[\![P]\!] \in \gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}[\![P]\!]$ (preuve de correction), ou
   – $\gamma(\mathcal{S}^\sharp[\![P]\!]) \not\subseteq \mathcal{Q}[\![P]\!]$ (propriété non satisfaite (erreur) ou approximation trop grossière (fausse alarme))

Difficulté de l'analyse statique

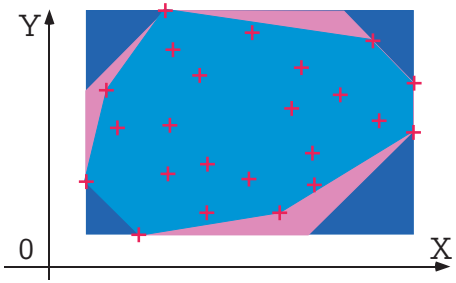
7. L'abstraction doit être choisie en fonction du programme P et de la propriété $\mathcal{Q}[\![P]\!]$ à démontrer
   – suffisamment grossière pour être calculable par un ordinateur,
   – suffisamment précise pour obtenir une preuve formelle de correction : $\gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}[\![P]\!]$;

Raffinement de l'analyse statique

8. En cas de fause alarme, il est nécessaire de raffiner l'abstraction.

Exemples d'abstraction

Domaines abstraits numériques d'usage général



Approximation d'un ensemble de points

Intervalles : [6]

$$\bigwedge_{i=1}^{n} a_i \leqslant x_i \leqslant b_i$$

Octogones :

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \pm x_i \pm y_j \leqslant a_{ij}$$

Polyèdres : [8]

$$\bigwedge_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ji} x_i\right) \leqslant b_j$$

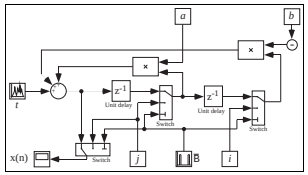
Exemple de raffinement de l'analyse statique

– Raffinement de l'analyse d'intervalles :
  – si $x \in [a, b]$ et $y \in [c, d]$ alors $x + y \in [a + c, b + d]$
  – Imprécis si $x = -y$ (il faudrait $x + y \in [0, 0]$)
  – raffinement nécessaire, ex. produit réduit avec $x = \pm y$ ou octogones [2]

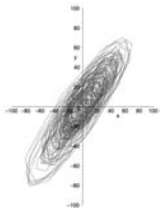
[2] Le raffinement par recherche de contre-exemples dans la sémantique concrète est évidemment illusoire à cause de l'explosion combinatoire !

Domaine abstrait ellipsoïdal

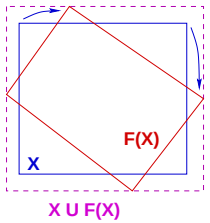
Filtre digital du 2ème ordre :



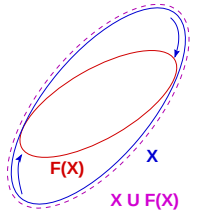
– Calcule $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
– Quelle abstraction peut démontrer que le calcul est borné ?
– Pas d'intervalle, octogone ou polyèdre stable
– Le volume le plus simple est un ellipsoïde [13]



trace d'exécution



intervalle instable



ellipsoïde stable

Exemple de filtre

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
             + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```


Divergences lentes par cumuls d'arrondis

```
X = 1.0;
while (TRUE) { ①
  X = X / 3.0;
  X = X * 3.0;
}
```

– Sur les réels $\mathbb{R}$: $x = 1.0$ en ①
– Sur les flottants : erreurs d'arrondi
– Cumul des erreurs d'arrondi : cause possible de divergence

Solution [17] : borner l'erreur d'arrondi cumulée en fonction du nombre d'itérés par des progressions arithmético-géométriques :

– Relation $|\mathtt{X}| \leq a \cdot b^n + c$, où a, b, c sont des constantes déterminées par l'analyse, n est le numéro d'itéré
– Nombre d'itérations borné par N : $|\mathtt{X}| \leq a \cdot b^N + c$

L'analyseur statique ASTRÉE

www.astree.ens.fr

Programmes analysés par ASTRÉE

– Domaine d'application: grands logiciels embarqués critiques de contrôle/commande temps-réel synchrone non linéaire de systèmes complexes.
– programmes C :
  – avec
    - types numériques de base, structures et tableaux
    - pointeurs (y compris sur des fonctions)
    - calculs flottants
    - tests, boucles et appels de fonctions
    - branchements limités (`goto` en avant, `break`, `continue`)

– avec (suite)
  – `union`
  – arithmétique de pointeurs
– sans
  – allocation dynamique de mémoire
  – fonctions récursives
  – branchements en arrière
  – effets de bord conflictuels
  – librairies C, appels système (parallélisme)

Spécifications implicites : absence d'erreurs à l'exécution

– Aucune violation de la norme de C (ex. débordement d'index de tableau, division par zéro)
– Aucun comportement indéfini dépendant de l'implémentation (ex. l'entier court maximal est 32767, NaN)
– Aucune violation des normes de programmation (ex. les variables statiques peuvent ne pas être initialisées à 0)
– Aucune violation des assertions du programmeur (les `assert` doivent être toutes vérifiées statiquement).

Aspects difficiles

– Taille : 100 à 1000 kLOCs, 10/50 000 variables globales
– Calculs flottants :
  y compris des réseaux interconnectés de filtres, du contrôle non-linéaire avec rétroactions, des interpolations...
– Nombreuses interdépendances entre variables :
  – La stabilité des calculs doit être prouvée
  – Des relations complexes doivent être inférées entre des valeurs booléennes et numériques
  – Il faut suivre de très longs chemins de calcul entre les entrées et les sorties

Caractéristiques de l'analyseur ASTRÉE

Statique : analyse à la compilation ($\neq$ analyse à l'exécution Rational Purify, Parasoft Insure++)

Analyseur de programmes : analyse des textes de programmes pas de micro-modèles de ces programmes ($\neq$ PROMELA dans SPIN ou Alloy dans l'Alloy Analyzer)

Automatique : pas d'interaction avec l'utilisateur ($\neq$ ESC Java, ESC Java 2)

Caractéristiques de l'analyseur ASTRÉE (suite)

Correct : couvre tout l'espace d'états accessibles ($\neq$ MAGIC, CBMC) et donc
- n'omet jamais une erreur potentielle ($\neq$ UNO, CMC de coverity.com)
- ne montre pas que les erreurs les plus probables ($\neq$ Splint),
- prend l'environnement (inconnu) en compte ($\neq$ SLAM)

Multiabstractions : utilise de nombreux domaines abstraits numériques et symboliques ($\neq$ constraintes symboliques de Bane ou l'abstraction canonique de TVLA)

Caractéristiques de l'analyseur ASTRÉE (suite)

Infinitaire : les abstractions utilisent des domaines abstraits infinis avec élargissement/rétrécissement ($\neq$ analyseurs basés sur le "model-checking" comme VeriSoft, Bandera, Java PathFinder)

Efficace : terminate toujours ($\neq$ abstraction/raffinement guidé par des contre-exemples BLAST, SLAM)

Spécialisable : peut facilement incorporer de nouvelles abstractions et les réductions entre domaines abstraits ($\neq$ analyseurs de portée générale comme PolySpace Verifier)

Caractéristiques de l'analyseur ASTRÉE (suite)

Orienté applications : a des connaissances de la théorie du contrôle/commande (ex. filtres digitaux) (à l'opposé d'une simple spécialisation à un style de programmation C Global Surveyor)

Paramétrique : le rapport coût/précision peut être ajusté par des options et directives dans le code pour satisfaire les besoins des utilisateurs

Paramétrisation automatique : la production des directives d'analyse dans le code peut être programmée (pour une spécialisation à un domaine d'application)

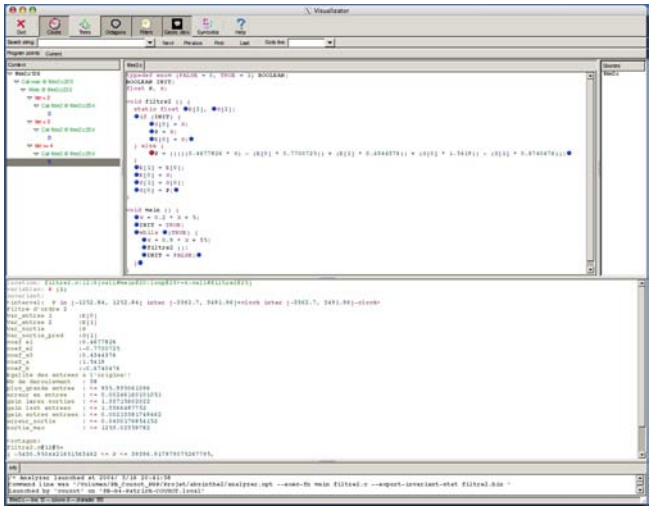
Caractéristiques de l'analyseur ASTRÉE (suite)

Modulaire : une instance de l'analyseur est construite par assembalge de modules O-CAML modules à partir d'une collection d'implémentations de domaines abstraits

Précis : peut ou pas de fausses alarmes par adaptation à un domaine d'application $\longrightarrow$

c'est un VÉRIFICATEUR AUTOMATIQUE !

Exemple de session d'analyse



Applications d'ASTRÉE

Application aux A 340/ A 380

– Logiciel primaire de contrôle de vol du système de commande de vol électrique de la famille des Airbus A340 et de l'A380

– programme C, automatiquement engendré à partir d'une spécification de haut niveau (à la Simulink/SCADE)

– A340 : 100.000 à 250.000 lignes

– A380 : 400.000 à 1.000.000 lignes

Une première mondiale

– En nov. 2005, analyse de 400.000 lignes de code C [3]

temps	mémoire	fausses alarmes
13h 52mn	2,2 Go	**0**

– En nov. 2006, analyse de 750.000 lignes de code C

temps	mémoire	fausses alarmes
34h 30mn	4,8 Go	**0**

[3] sur un AMD Opteron 248, 64 bits, un seul processeur

Conclusion

Résultats du projet ASTRÉE

– Preuve pratique que l'analyse statique par interprétation abstraite passe à l'échelle [4]
– Utilisation industrielle effective

[4] La principale difficulté pour les méthodes formelles.

Projet THÉSÉE

– Analyse statique de programmes asynchrones
– Absence d'erreurs à l'exécution, d'interférences incorrectes et de mauvaises synchronisations
– Projet Airbus France, CNRS, EDF, ENS
– Exemple d'application : A380 Flight Warning System sur ARINC 653 (3 500 000 Locs)

FIN, MERCI DE VOTRE ATTENTION

Plus de références à l'URL www.di.ens.fr/~cousot
www.astree.ens.fr.

Références introductives sur l'interprétation abstraite

[1] Patrick Cousot.
Interprétation abstraite.
Technique et Science Informatique, Vol. 19, Nb 1-2-3. Janvier 2000, Hermès, Paris, France. pp. 155—164.

[2] Patrick Cousot.
Abstract Interpretation Based Formal Methods and Future Challenges.
In Informatics, 10 Years Back - 10 Years Ahead, R. Wilhelm (Ed.), Lecture Notes in Computer Science 2000, pp. 138—156, 2001.

[3] Patrick Cousot & Radhia Cousot.
Basic Concepts of Abstract Interpretation.
In *Building the Information Society*, R. Jacquard (Ed.), Kluwer Academic Publishers, pp. 359—366, 2004.

[4] Patrick Cousot.
Abstract Interpretation.
http://www.di.ens.fr/~cousot/AI/

Références classiques sur l'interprétation abstraite

[5] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.

[6] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. Second International Symp. on Programming*, pp. 106–130, 1976. Dunod, Paris, France.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conf. Rec. Fourth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pp. 238–252, Los Angeles, California, 1977. ACM Press, New York, USA.

[8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a pro- gram. In *Conf. Rec. Fifth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pp. 84–97, Tucson, Arizona, 1978. ACM Press, New York, U.S.A.

[9] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conf. Rec. Sixth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pp. 269–282, San Antonio, Texas, 1979. ACM Press, New York, U.S.A.

Références sur ASTRÉE

[10] ASTRÉE, http://www.astree.ens.fr/

[11] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux & Xavier Rival.
Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software, invited chapter. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, T. Mogensen and D.A. Schmidt and I.H. Sudborough (Editors). Lecture Notes in Computer Science 2566, pp. 85—108, © Springer.

[12] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, & Xavier Rival.
A Static Analyzer for Large Safety-Critical Software. In PLDI 2003 — ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation, 2003 Federated Computing Research Conference, June 7—14, 2003, San Diego, California, USA, pp. 196—207, © ACM.

[13] Jérôme Feret.
Static analysis of digital filters. In ESOP 2004 — European Symposium on Programming, D. Schmidt (editor), Mar. 27 —Apr. 4, 2004, Barcelona, ES, Lecture Notes in Computer Science 2986, pp. 33—48, © Springer.

[14] Laurent Mauborgne.
ASTRÉE: verification of absence of run-time error. In *Building the Information Society*, R. Jacquard (Ed.), Kluwer Academic Publishers, pp. 385—392, 2004.

[15] Antoine Miné.
Relational abstract domains for the detection of floating-point run-time errors. In ESOP 2004 — European Symposium on Programming, D. Schmidt (editor), Mar. 27 — Apr. 4, 2004, Barcelona, Lecture Notes in Computer Science 2986, pp. 3—17, © Springer.

[16] Antoine Miné.
Weakly relational numerical abstract domains. Thèse de l'École polytechnique, 6 December 2004.

[17] Jérôme Feret.
The arithmetic-geometric progression abstract domain. In VMCAI 2005 — Verification, Model Checking and Abstract Interpretation, R. Cousot (editor), Lecture Notes in Computer Science 3385, pp. 42—58, 17—19 January 2005, Paris, © Springer.

[18] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux & Xavier Rival.
The ASTRÉE analyser. In ESOP 2005 — The European Symposium on Programming, M. Sagiv (editor), Lecture Notes in Computer Science 3444, pp. 21—30, 2—10 April 2005, Edinburgh, © Springer.

[19] Laurent Mauborgne & Xavier Rival.
Trace Partitioning in Abstract Interpretation Based Static Analyzer. In ESOP 2005 — ; The European Symposium on Programming, M. Sagiv (editor), Lecture Notes in Computer Science 3444, pp. 5—20, 2—10 April 2005, Edinburgh, © Springer.

[20] Xavier Rival.
Understanding the Origin of Alarms in ASTRÉE. In SAS'05 — The 12th International Static Analysis Symposium, Chris Hankin & Igor Siveroni (editors), Lecture Notes in Computer Science 3672, pp. 303—319, 7—9 September 2005, London, UK, © Springer.

[21] David Monniaux.
The Parallel Implementation of the Astree Static Analyzer. In APLAS 2005 — The Third Asian Symposium on Programming Languages and Systems, Kwangkeun Yi (editor), Lecture Notes in Computer Science 3780, pp. 86—96, 2—5 November 2005, Tsukuba, Japan, © Springer.

[22] Xavier Rival.
Abstract Dependences for Alarm Diagnosis. In APLAS 2005 — The Third Asian Symposium on Programming Languages and Systems, Kwangkeun Yi (editor), Lecture Notes in Computer Science 3780, pp. 347—363, 2—5 November 2005, Tsukuba, Japan, © Springer.

[23] Antoine Miné.
Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In VMCAI 2006 — Seventh International Conference on Verification, Model Checking and Abstract Interpretation, E. Allen Emerson & Kedar S. Namjoshi (editors), Lecture Notes in Computer Science 3855, pp. 348—363, 8—10 January 2006, Charleston, South Carolina, USA, © Springer.

[24] Antoine Miné.
Field-Sensitive Value Analysis of Embedded C Programs with Union Types and Pointer Arithmetics. In Proceedings of the 2006 ACM SIGPLAN/SIGBED Conference for Languages, Compilers, and Tools for Embedded Systems (LCTES 2006), 14—16 June 2006, Ottawa, Ontario, Canada. ACM Press, pp. 54—63.