

# La vérification des programmes par interprétation abstraite

Patrick Cousot

École normale supérieure

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

Équipe-projet INRIA Paris-Rocquencourt/CNRS/ENS « ABSTRACTION »

Séminaire

Chaire d'innovation technologique Liliane Bettencourt

Collège de France, 22 février 2008



Pourquoi et comment le monde devient numérique, 22/2/2008

— 1 —

© P. Cousot 

## 1. Exemples classiques de bugs



Pourquoi et comment le monde devient numérique, 22/2/2008

— 2 —

© P. Cousot 

## Exemples classiques de bugs du calcul en entiers

### Le programme factorielle (fact.c)

```
#include <stdio.h>
int fact (int n) {
    int r, i;
    r = 1;
    for (i=2; i<=n; i++) {
        r = r*i;
    }
    return r;
}
int main() { int n;
    scanf("%d",&n);
    printf("%d != %d\n", n, fact(n));
}
```

$\leftarrow \text{fact}(n) = 2 \times 3 \times \dots \times n$

$\leftarrow$  lire  $n$  (tapé au clavier)

$\leftarrow$  écrire  $n! = \text{fact}(n)$



Pourquoi et comment le monde devient numérique, 22/2/2008

— 3 —

© P. Cousot 



Pourquoi et comment le monde devient numérique, 22/2/2008

— 4 —

© P. Cousot 

## Compilation<sup>(1)</sup> du programme factorielle (fact.c)

```
#include <stdio.h>
int fact (int n) {
    int r, i;
    r = 1;
    for (i=2; i<=n; i++) {
        r = r*i;
    }
    return r;
}

int main() { int n;
    scanf("%d",&n);
    printf("%d != %d\n",n,fact(n));
}
```

(1) Voir la leçon du 8 février 2008 et le séminaire de Xavier Leroy

## Exécutions du programme factorielle (fact.c)

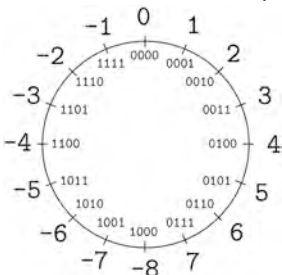
```
#include <stdio.h>
int fact (int n) {
    int r, i;
    r = 1;
    for (i=2; i<=n; i++) {
        r = r*i;
    }
    return r;
}

int main() { int n;
    scanf("%d",&n);
    printf("%d != %d\n",n,fact(n));
}
```

3  
3 ! = 6  
4  
4 ! = 24  
100  
100 ! = 0  
20  
20 ! = -2102132736

## À la chasse au bug

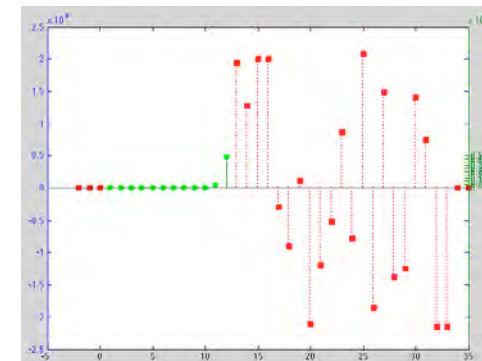
- Les ordinateurs utilisent une **arithmétique entière modulaire** sur  $n$  bits (où  $n = 16, 32, 64$ , etc)
- Exemple d'une **représentation des entiers sur 4 bits** (en **complément à deux**) :



- Seuls les entiers entre -8 et 7 sont représentés sur 4 bits
- On obtient  $7 + 2 = -7$   
 $7 + 9 = 0$

## Le bug est une défaillance du programmeur

En machine, la fonction `fact(n)` ne coïncide avec  $n! = 2 \times 3 \times \dots \times n$  sur les entiers que pour  $1 \leq n \leq 12$  :



Et en OCAML on a un résultat différent !

```
let rec fact n = if (n = 1) then 1 else n * fact(n-1);;
```

fact(n)	C	OCAML	fact(22)	-522715136	-522715136
fact(1)	1	1	fact(23)	862453760	862453760
...	...	...	fact(24)	-775946240	-775946240
fact(12)	479001600	479001600	fact(25)	2076180480	-71303168
fact(13)	1932053504	-215430144	fact(26)	-1853882368	293601280
fact(14)	1278945280	-868538368	fact(27)	1484783616	-662700032
fact(15)	2004310016	-143173632	fact(28)	-1375731712	771751936
fact(16)	2004189184	-143294464	fact(29)	-1241513984	905969664
fact(17)	-288522240	-288522240	fact(30)	1409286144	-738197504
fact(18)	-898433024	-898433024	fact(31)	738197504	738197504
fact(19)	109641728	109641728	fact(32)	-2147483648	0
fact(20)	-2102132736	45350912	fact(33)	-2147483648	0
fact(21)	-1195114496	952369152	fact(34)	0	0

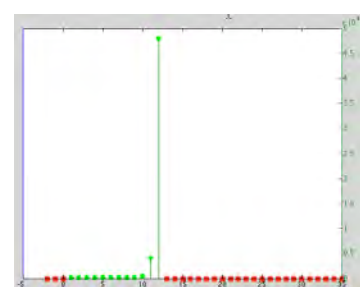
Pourquoi ? Que fait fact(-1) ?

Preuve d'absence d'erreurs à l'exécution par analyse statique

```
% cat -n fact_lim.c
1 int MAXINT = 2147483647;
2 int fact (int n) {
3     int r, i;
4     if (n < 1) || (n = MAXINT) {
5         r = 0;
6     } else {
7         r = 1;
8         for (i = 2; i<=n; i++) {
9             if (r <= (MAXINT / i)) {
10                 r = r * i;
11             } else {
12                 r = 0;
13             }
14         }
15     }
16     return r;
17 }
18
```

```
19 int main() {
20     int n, f;
21     f = fact(n);
22 }
```

% astree -exec-fn main fact\_lim.c |& grep WARN  
%  
→ Aucune alarme !



Exemples classiques de bugs  
du calcul en flottants

Les modèles et leur réalisation sur machine

- Les **modélisations mathématiques** des systèmes physiques utilisent les **nombre réels**
- Les **langages informatiques de modélisation** (comme SCAD<sup>(2)</sup>) utilisent les **nombre réels**
- Les **nombre réels** sont difficilement représentables en machine ( $\pi$  a un nombre infini de décimales)
- Les **langages informatiques de programmation** (comme C ou OCAML) utilisent les **nombre flottants**

<sup>(2)</sup> Voir la leçon du 15 février 2008

## Les flottants

- Les *nombres flottants* sont un sous-ensemble des *rationnels*
- Par exemple on peut représenter 32 flottants sur 6 bits, les 16 flottants positifs étant répartis comme suit :



- Quand les calculs réels ne tombent pas juste, il faut *arrondir vers un flottant proche*

## Exemple d'erreur d'arrondi (1)

$$(x + a) - (x - a) \neq 2a$$

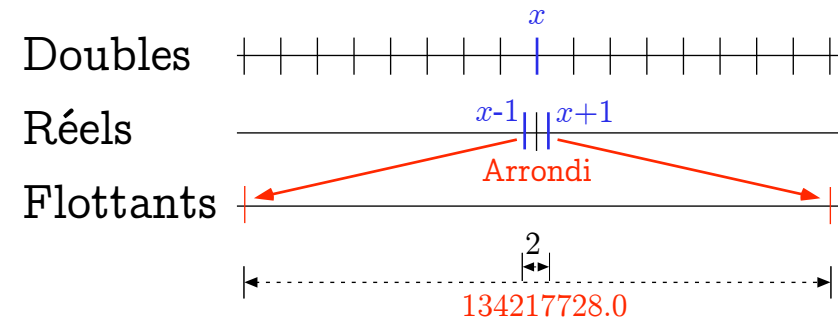
```
#include <stdio.h>           % gcc arrondi1.c -o arrondi1.exec
int main() {                 % ./arrondi1.exec
    double x, a; float y, z; 134217728.000000
    x = 1125899973951488.0; %
    a = 1.0;
    y = (x+a);
    z = (x-a);
    printf("%f\n", y-z);
}
```

## Exemple d'erreur d'arrondi (2)

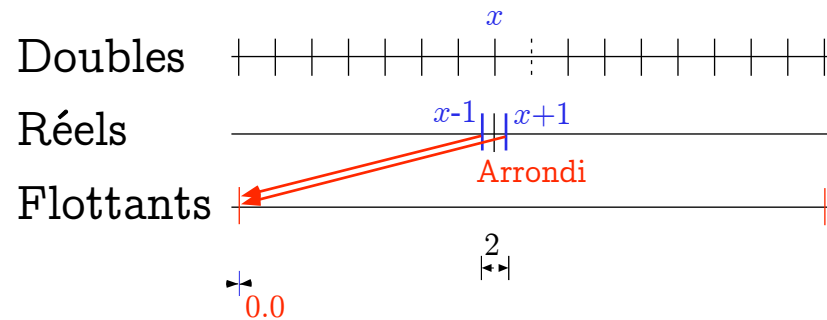
$$(x + a) - (x - a) \neq 2a$$

```
#include <stdio.h>           % gcc arrondi2.c -o arrondi2.exec
int main() {                 % ./arrondi2.exec
    double x, a; float y, z; 0.000000
    x = 1125899973951487.0; %
    a = 1.0;
    y = (x+a);
    z = (x-a);
    printf("%f\n", y-z);
}
```

## À la chasse au bug (1)



## À la chasse au bug (2)



## Preuve d'absence d'erreurs à l'exécution par analyse statique

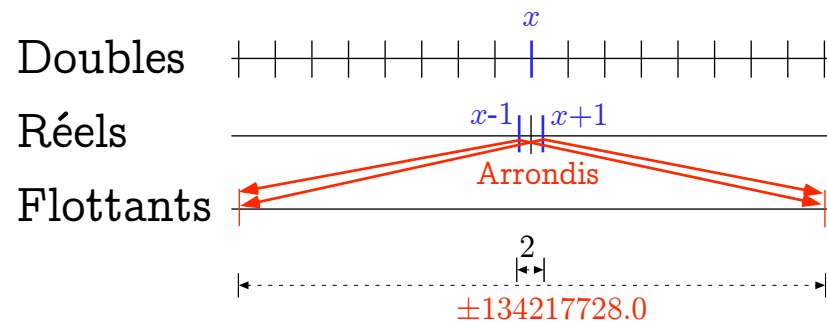
```
% cat -n arrondi3.c
1 int main() {
2     double x; float y, z, r;;
3     x = 1125899973951488.0;
4     y = x + 1;
5     z = x - 1;
6     r = y - z;
7     __ASTREE_log_vars((r));
8 }

% astree -exec-fn main -print-float-digits 10 arrondi3.c \
|& grep "r in "

direct = <float-interval : r in [-134217728, 134217728] >(3)
```

<sup>(3)</sup> ASTRÉE considère le pire des cas d'arrondi (vers  $+\infty$ ,  $-\infty$ , 0 ou au plus proche) d'où la possibilité d'obtenir -134217728.

## Vérification faite dans le pire des cas



## Exemples de bugs dus à des erreurs d'arrondi

- Le **bug du missile patriote** ratant les Scuds en 1991 à cause une horloge incrémentée par  $\frac{1}{10}$ ème de seconde ( $((0,1)_{10} = (0,0001100110011001100\dots)_2$  en binaire)
- Le **bug d'Exel 2007** :  $77,1 \times 850$  qui donne 65.535 mais s'affiche en 100.000! <sup>(4)</sup>

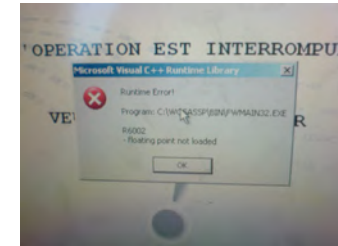
2	65535-2 <sup>^</sup> (-37)	100000	65536-2 <sup>^</sup> (-37)	100001
3	65535-2 <sup>^</sup> (-36)	100000	65536-2 <sup>^</sup> (-36)	100001
4	65535-2 <sup>^</sup> (-35)	100000	65536-2 <sup>^</sup> (-35)	100001
5	65535-2 <sup>^</sup> (-34)	65535	65536-2 <sup>^</sup> (-34)	65536
6	65535-2 <sup>^</sup> (-36)-2 <sup>^</sup> (-37)	100000	65536-2 <sup>^</sup> (-36)-2 <sup>^</sup> (-37)	100001
7	65535-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-37)	100000	65536-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-37)	100001
8	65535-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-36)	100000	65536-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-36)	100001
9	65535-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-36)-2 <sup>^</sup> (-37)	65535	65536-2 <sup>^</sup> (-35)-2 <sup>^</sup> (-36)-2 <sup>^</sup> (-37)	65536

<sup>(4)</sup> Erreur d'arrondi incorrect lors de la traduction de flottants IEEE 754 sur 64 bits en chaîne de caractères Unicode qui conduit à un mauvais alignement dans une table de conversion. Le bug apparaît exactement pour six nombres entre 65534.99999999995 et 65535 et six entre 65535.99999999995 et 65536.

## Les bugs dans le monde numérisé quotidien

### Les bugs sont fréquents dans la vie quotidienne

- Les **bugs** se trouvent dans les banques, les voitures, les téléphones, les machines à laver, ...
- Exemple (**bug dans un distributeur de monnaie** au 19 Boulevard Sébastopol à Paris, le 21 novembre 2006 à 8<sup>h</sup>30) :



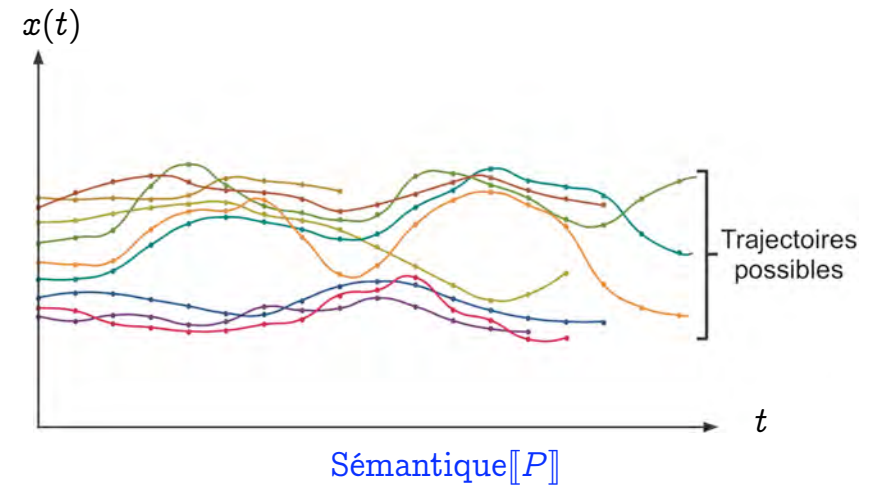
## 2. La vérification des programmes

### Principe de la vérification des programmes

- Définir une **sémantique** du langage (c'est-à-dire l'effet de l'exécution des programmes du langage)
- Définir une **spécification** (exemple : pas d'erreur à l'exécution comme une division par zéro, un débordement arithmétique, etc)
- Faire une **preuve formelle** que la sémantique satisfait la spécification
- Utiliser l'ordinateur pour **automatiser la preuve**

## Sémantique des programmes

## Sémantique opérationnelle du programme $P$



## Exemple : trace d'exécution de $\text{fact}(4)$ <sup>(5)</sup>

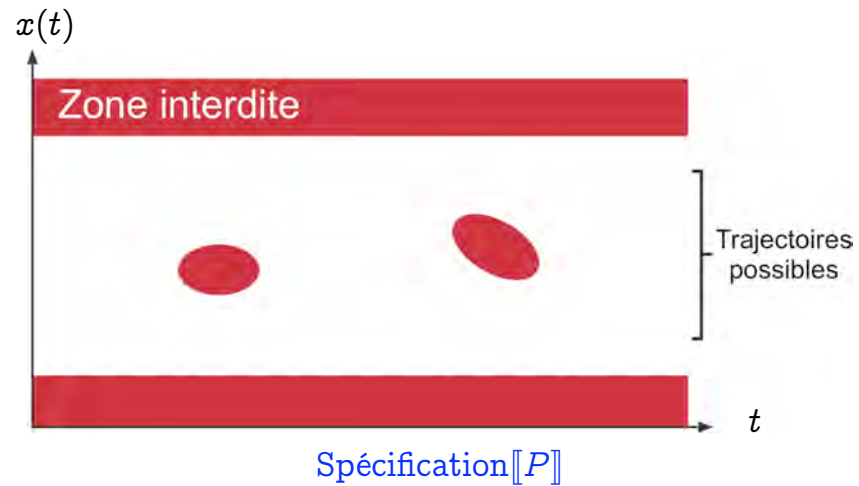
```
int fact (int n ) {  
  int r = 1, i;  
  for (i=2; i<=n; i++) {  
    r = r*i;  
  }  
  return r;  
}
```

•  $n \leftarrow 4; r \leftarrow 1;$   
•  $i \leftarrow 2; r \leftarrow 1 \times 2 = 2;$   
•  $i \leftarrow 3; r \leftarrow 2 \times 3 = 6;$   
•  $i \leftarrow 4; r \leftarrow 6 \times 4 = 24;$   
•  $i \leftarrow 5;$   
•  $\text{return } 24;$

<sup>(5)</sup> Voir la leçon du 22 février 2008

## Spécification des programmes

## Spécification du programme $P$



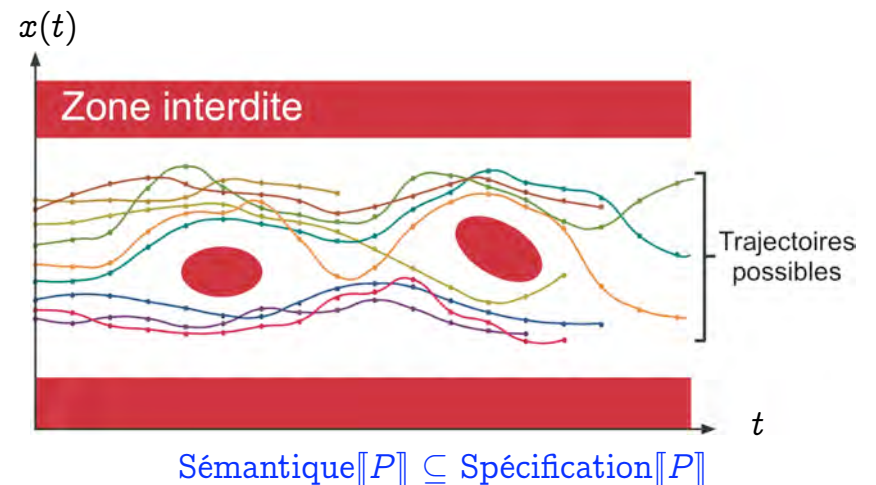
## Exemple de spécification

```
int fact (int n) {
    int r, i;
    r = 1;
    for (i=2; i<=n; i++) {
        r = r*i;
    }
    return r;
}
```

← pas de débordement de  $i++$   
← pas de débordement de  $r*i$

## Preuve formelle

## Preuve formelle du programme $P$



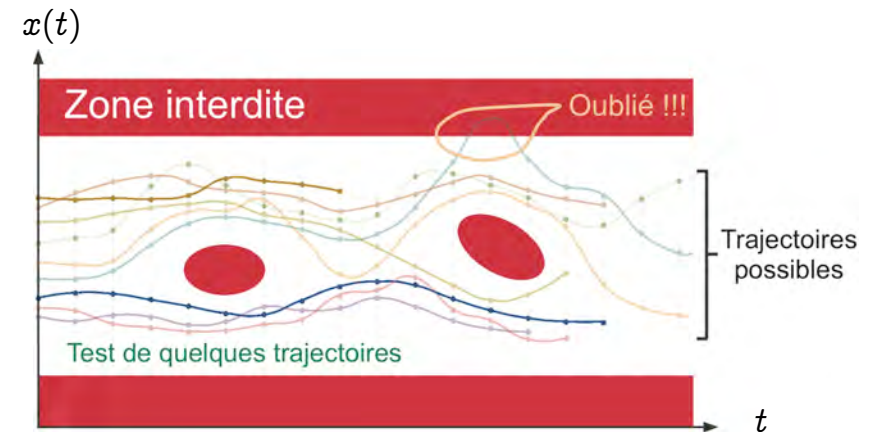


## Indécidabilité et complexité

- Le problème de la preuve mathématique formelle est **indécidable** <sup>(6)</sup>
- Même en supposant tout fini, la **complexité** est beaucoup trop élevée
- Exemple : 1.000.000 lignes  $\times$  50.000 variables  $\times$  64 bits  $\simeq 10^{27}$  états
- À raison de l'examen de  $10^{15}$  états par seconde, il faudrait  $10^{12}$  s  $>$  300 siècles (et beaucoup de mémoire) !

<sup>(6)</sup> un ordinateur ne peut pas toujours le résoudre en un temps fini, voir la leçon du 22 février 2008

## Le test est incomplet

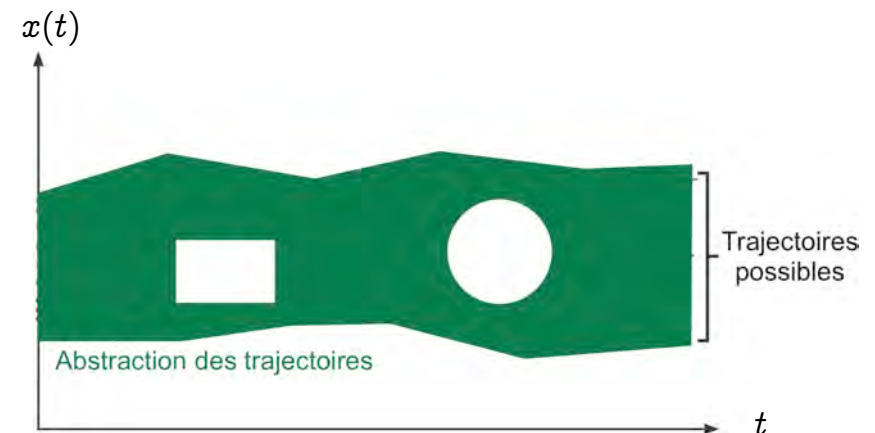


## 3. Interprétation abstraite [1]

### Référence

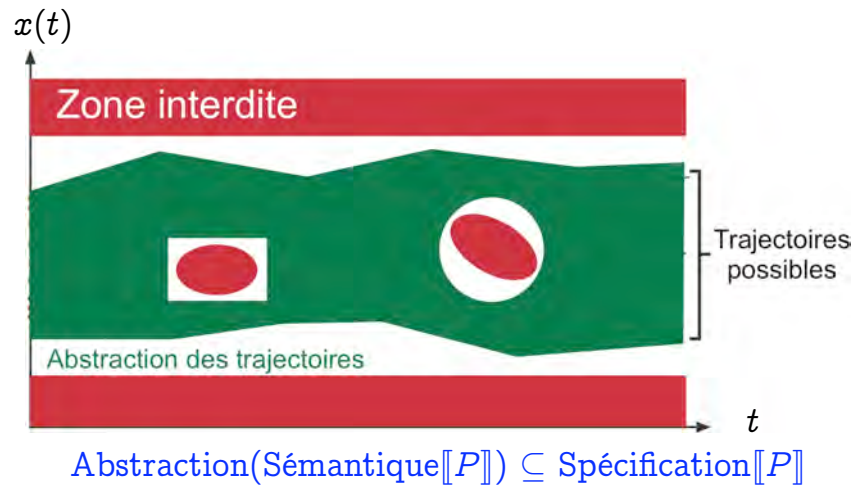
- [1] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse d'Etat ès sciences mathématiques. Université scientifique et médicale de Grenoble. 1978.

## Abstraction du programme $P$



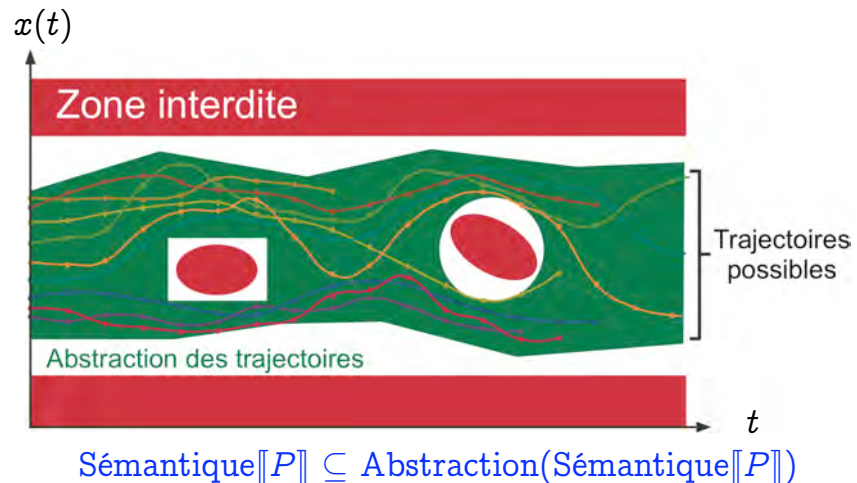
$\text{Abstraction}(\text{Sémantique}[P])$

## Preuve par abstraction

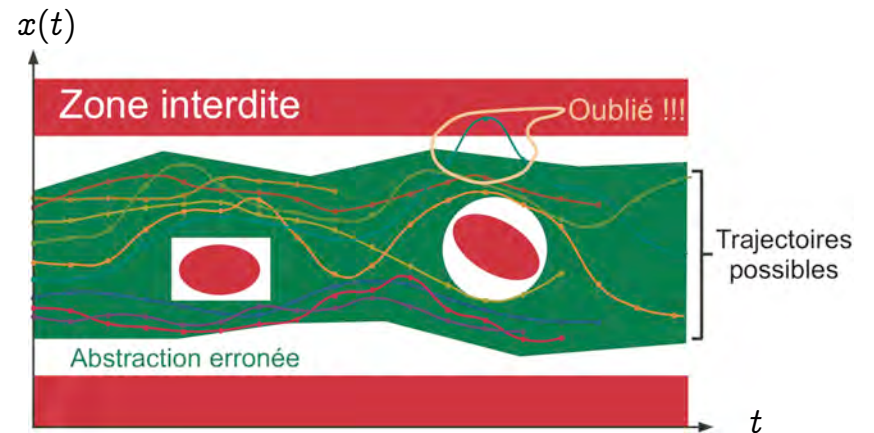


## Correction de l'interprétation abstraite

## L'interprétation abstraite est correcte



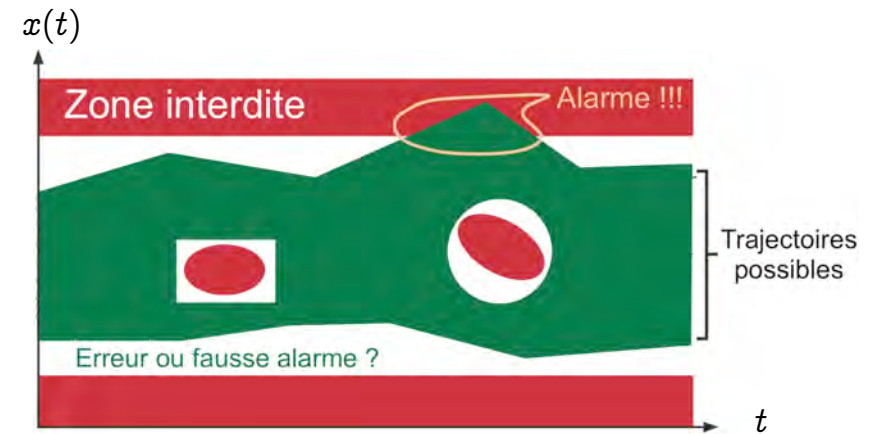
## Exemple d'abstraction erronée <sup>(7)</sup>



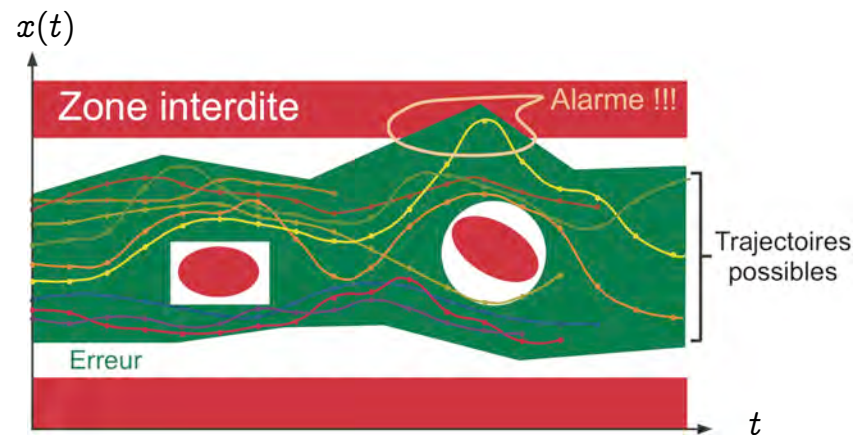
<sup>(7)</sup> Cette situation est toujours exclue par la théorie de l'interprétation abstraite.

## Incomplétude de l'interprétation abstraite

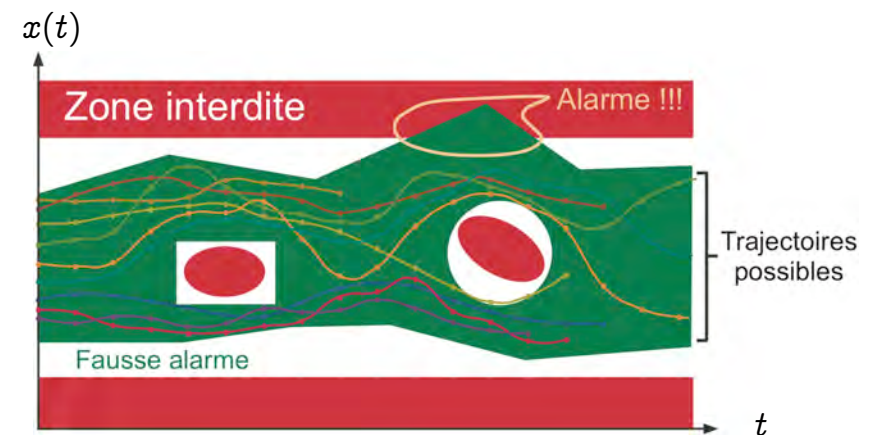
## Alarme



## Une alarme peut correspondre à une erreur



## Une alarme peut correspondre à une approximation



## Applications de l'interprétation abstraite vues dans le cours

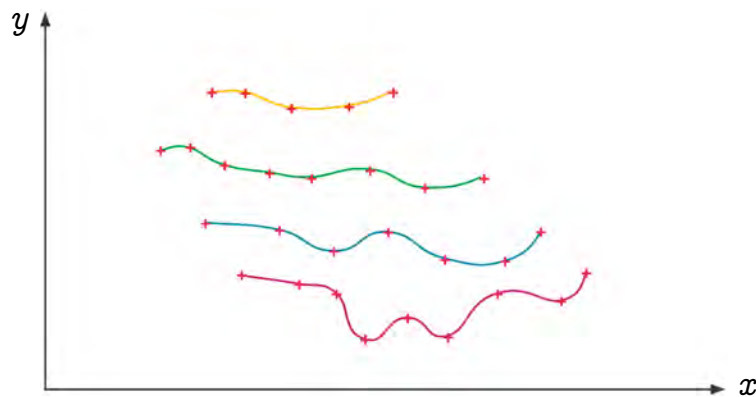
- Le **typage** <sup>(8)</sup> [Cou97]
- Le “**model-checking**” <sup>(9)</sup> [CC00]
- La **transformation des programmes** (par exemple pour l'optimisation <sup>(8)</sup> lors de la compilation) [CC02]
- La définition de **sémantiques** à différents niveaux d'abstraction [Cou02]
- L'**analyse statique** pour vérifier l'absence de bugs [BCC<sup>+</sup>03]
- ...

<sup>(8)</sup> Voir la leçon du 8 février 2008 et le séminaire de Xavier Leroy

<sup>(9)</sup> Voir la leçon du 22 février 2008

## 4. Application de l'interprétation abstraite à l'analyse statique

### Sémantique



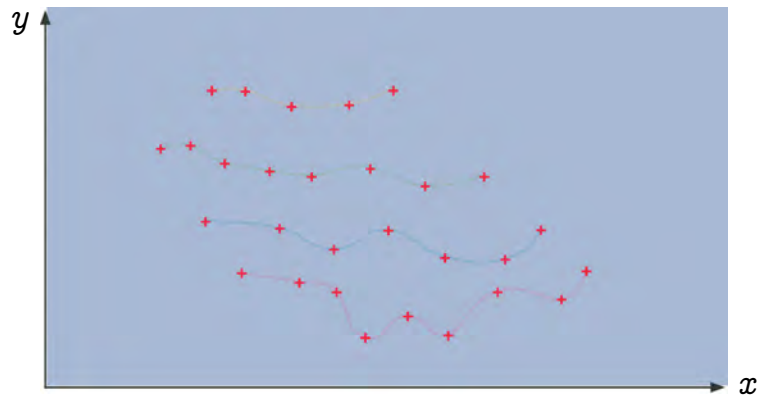
Ensemble (infini) de traces (finies ou infinies)

### Abstraction en un ensemble d'états (invariant)



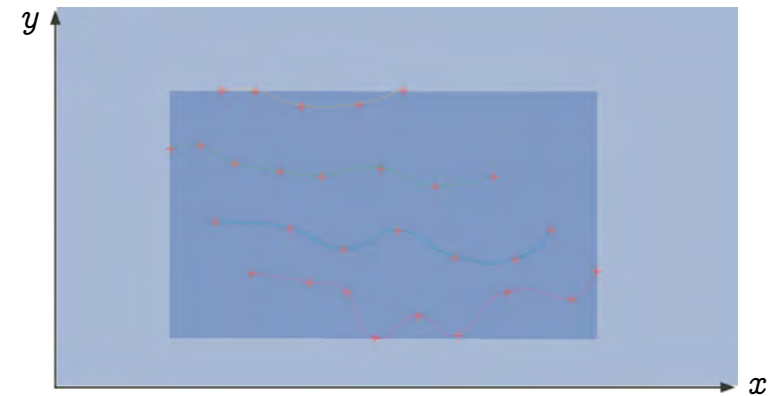
Ensemble de points  $\{(x_i, y_i) : i \in \Delta\}$ , Hoare logic [Cou02]

### Abstraction par des signes



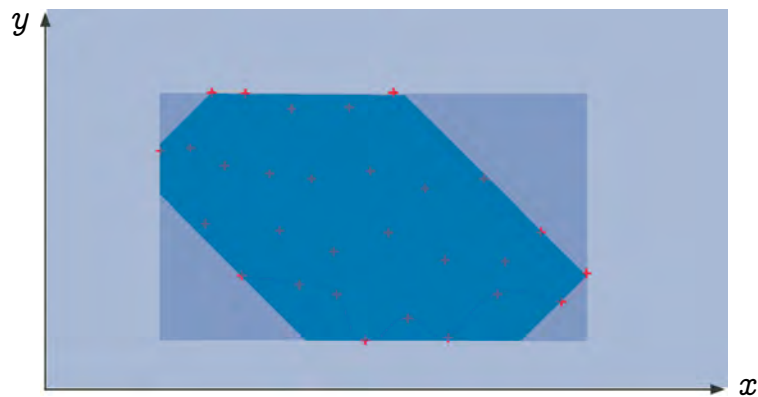
Signes  $x \geq 0, y \geq 0$  [CC79]

### Abstraction par des intervalles



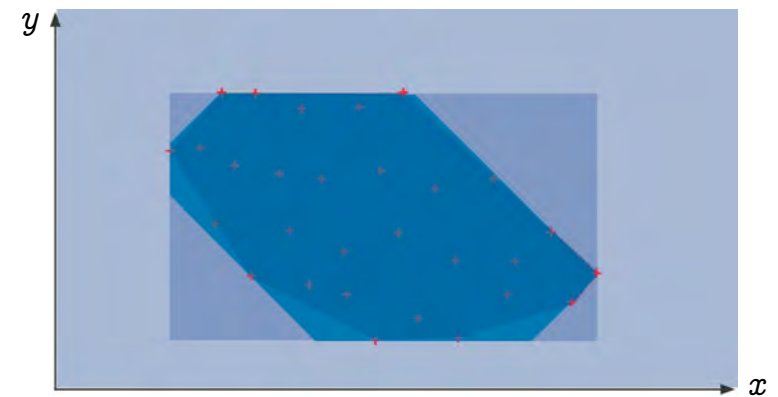
Intervalles  $a \leq x \leq b, c \leq y \leq d$  [CC77]

### Abstraction par des octogones



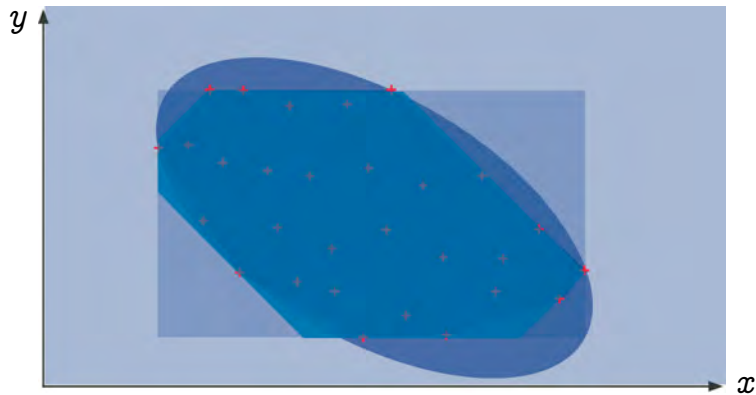
Octogones  $x - y \leq a, x + y \leq b$  [Min06]

### Abstraction par des polyèdres



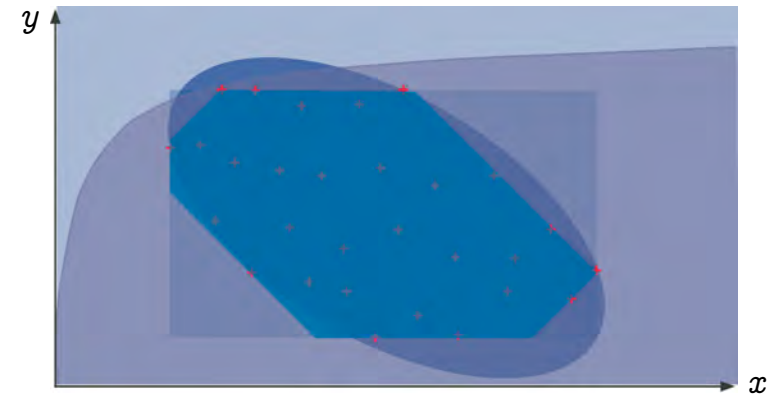
Polyèdres  $a.x + b.y \leq c$  [CH78]

## Abstraction par des ellipsoïdes



Ellipsoïdes  $(x - a)^2 + (y - b)^2 \leq c$  [Fer05b]

## Abstraction par des exponentielles



Exponentielles  $a^x \leq y$  [Fer05a]

## 5. Calcul d'invariant par approximation de points fixes [CC77]

### Équation de point fixe

```

{y ≥ 0} ← hypothèse
x = y
{I(x, y)} ← invariant de boucle
while (x > 0) {
  x = x - 1;
}
    
```

Conditions de vérification de Floyd-Hoare :

$$(y \geq 0 \wedge x = y) \implies I(x, y)$$

*initialisation*

$$(I(x, y) \wedge x > 0 \wedge x' = x - 1) \implies I(x', y)$$

*iteration*

Équation de point fixe :

$$I(x, y) = x \geq 0 \wedge (x = y \vee I(x + 1, y)) \quad (\text{i.e. } I = F(I)^{(10)})$$

<sup>(10)</sup> On cherche l'invariant  $I$  le plus précis, qui implique tous les autres



Itérés  $I = \lim_{n \rightarrow \infty} F^n(\text{faux}) \dots$  accélérés

$$I^0(x, y) = \text{faux}$$

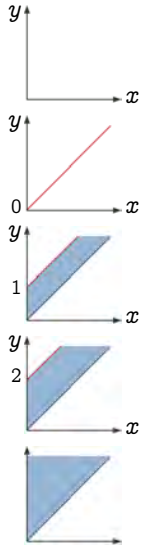
$$I^1(x, y) = x \geq 0 \wedge (x = y \vee I^0(x + 1, y)) \\ = 0 \leq x = y$$

$$I^2(x, y) = x \geq 0 \wedge (x = y \vee I^1(x + 1, y)) \\ = 0 \leq x \leq y \leq x + 1$$

$$I^3(x, y) = x \geq 0 \wedge (x = y \vee I^2(x + 1, y)) \\ = 0 \leq x \leq y \leq x + 2$$

$$I^4(x, y) = I^2(x, y) \nabla I^3(x, y) \leftarrow \text{widening} \\ = 0 \leq x \leq y$$

$$I^5(x, y) = x \geq 0 \wedge (x = y \vee I^4(x + 1, y)) \\ = I^4(x, y) \text{ point fixe!}$$



## 6. Application industrielle de l'interprétation abstraite

### Exemples d'analyseurs statiques

- Pour les programmes C critiques synchrones embarqués de contrôle/commande (par exemple pour des logiciels de Commande De Vol Électrique<sup>(11)</sup>)
- aiT [FHL<sup>+</sup>01] est un analyseur statique qui vérifie les temps d'exécution maximaux (pour garantir la synchronisation en temps voulu)
- ASTRÉE [BCC<sup>+</sup>03] est un analyseur statique qui vérifie l'absence d'erreurs à l'exécution



<sup>(11)</sup> Voir la leçon du 15 février 2008 et le séminaire de Gérard Ladiet

### Exemple d'analyse par ASTRÉE

```
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; }
}
```

## Exemple d'analyse par ASTRÉE

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P = (P - (((2.0 * P) - A) - B)
      * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}

void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock();
  }
}

% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
|P| <= (15. + 5.87747175411e-39
/ 1.19209290217e-07) * (1 +
1.19209290217e-07)^clock -
5.87747175411e-39 / 1.19209290217e-07
<= 23.0393526881
```



## Résultats industriels obtenus avec ASTRÉE

Preuves automatiques d'absence  
d'erreur à l'exécution dans des  
logiciels de Commande De Vol  
Électrique<sup>(12)</sup> :



- Logiciel 1 : 132.000 lignes de C, 40mn sur un PC 2.8 GHz, 300 mégaoctets (nov. 2003)
- Logiciel 2 : 1.000.000 de lignes de C, 34h, 8 gigaoctets (nov. 2005)

sans aucune fausse alarme

Premières mondiales !

(12) Voir la leçon du 15 février 2008 et le séminaire de Gérard Ladier



## 7. Conclusion

## Conclusion

- **Vision** : pour comprendre le monde numérique, il faut l'analyser à différents niveaux d'abstraction
- **Théorie** : l'interprétation abstraite assure une cohérence entre ces abstractions et offre des techniques d'approximation effectives
- **Applications** : le choix d'abstractions effectives suffisamment grossières pour être calculables et précises pour éviter les fausses alarmes permet de vaincre l'indécidabilité et la complexité dans la vérification des modèles et des programmes





## Le futur

- **Génie informatique** : La vérification manuelle par contrôle de la méthode de conception du programme<sup>(13)</sup> sera complétée par la vérification automatique du programme produit
- **Systèmes complexes** : l'interprétation abstraite s'applique aussi bien à l'analyse des systèmes évolutifs dont on sait décrire le comportement discret (traitement d'images, systèmes biologiques, calcul quantique, etc)

(13) Voir le séminaire de Gérard Ladier du 15 février 2008

# FIN

## Merci de votre attention

## 8. Bibliographie

## Courte bibliographie

- [BCC<sup>+</sup>03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN '2003 Conference on Programming Language Design and Implementation (PLDI)*, pages 196–207, San Diego, California, USA, 7–14 juin 2003. ACM Press, New York, New York, USA.
- [CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, New York, USA.
- [CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, New York, USA.
- [CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Massachusetts, USA, janvier 2000. ACM Press, New York, New York, USA.
- [CC02] P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, USA, janvier 2002. ACM Press, New York, New York, USA.

- [CCF<sup>+</sup>07] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Varieties of static analyzers : A comparison with ASTRÉE, papier invité. In M. Hinchey, He Jifeng, and J. Sanders, editors, *Proceedings of the First IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE '07*, pages 3–17, Shanghai, Chine, 6–8 juin 2007. IEEE Computer Society Press, Los Alamitos, Californie, USA.
- [CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, New York, USA.
- [Cou97] P. Cousot. Types as abstract interpretations, papier invité. In *Conference Record of the Twenty-fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–331, Paris, janvier 1997. ACM Press, New York, New York, USA.
- [Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1–2) :47–103, 2002.
- [DS07] D. Delmas and J. Souyris. ASTRÉE : from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proceedings of the Fourteenth International Symposium on Static Analysis, SAS '07*, Kongens Lyngby, Danemark, Lecture Notes in Computer Science 4634, pages 437–451. Springer, Berlin, Allemagne, 22–24 août 2007.
- [Fer05a] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proceedings of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2005)*, pages 42–58, Paris, 17–19 janvier 2005. Lecture Notes in Computer Science 3385, Springer, Berlin, Allemagne.

- [Fer05b] J. Feret. Numerical abstract domains for digital filters. In *First International Workshop on Numerical & Symbolic Abstract Domains, NSAD '05*, Maison Des Polytechniciens, Paris, 21 janvier 2005.
- [FHL<sup>+</sup>01] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In T.A. Henzinger and C.M. Kirsch, editors, *Proceedings of the First International Workshop on Embedded Software, EMSOFT '2001*, volume 2211 of *Lecture Notes in Computer Science*, pages 469–485. Springer, Berlin, Allemagne, 2001.
- [Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19 :31–100, 2006.

## Réponses aux questions

- Les entiers sont codés sur 32 bits en C et 31 bits en OCAML (un bit étant gardé pour gérer la mémoire pour le ramassage de miettes (*garbage collection*))
- L'appel de `fact(-1)` appelle `fact(-2)` qui appelle `fact(-3)`, etc. À chaque appel, il faut empiler la paramètre et l'adresse de retour<sup>(14)</sup>, ce qui se termine par un débordement de la pile :

```
% ocaml
Objective Caml version 3.10.0
# let rec fact n = if (n = 1) then 1 else n * fact(n-1);;
val fact : int -> int = <fun>
# fact(-1);;
Stack overflow during evaluation (looping recursion?).
```

<sup>(14)</sup> Voir la leçon du 8 février 2008 et le séminaire de Xavier Leroy