

Analyse statique de code C

ESIR 2 – 10 janvier 2012

David MENTRÉ – dmentre@linux-france.org

Plan de la présentation

- ▶ **Prélude**
 - ▶ Quelques exemples introductifs
- ▶ **Analyse abstraite**
 - ▶ Qu'est-ce que c'est ? Comment ça marche ?
- ▶ **Analyse abstraite de code C avec Frama-C**
 - ▶ Analyse de valeurs
 - ▶ Génération d'alarmes
- ▶ **Conclusion**





Prélude : quelques exemples
illustratifs

Ce code contient une erreur !

- ▶ Calcul de la valeur absolue d'un nombre en langage C

```
int z_abs_x(const int x)
{
    int z;
    if (x < 0)
        z = -x;
    else /* x >= 0 */
        z = x;
    return z;
}
```



Ce code contient une erreur !

- ▶ Calcul de la valeur absolue d'un nombre en langage C

```
int z_abs_x(const int x)
{
    int z;
    if (x < 0)
        z = -x;
    else /* x >= 0 */
        z = x;
    return z;
}
```

- ▶ Solution : si $x = -2^{31}$?
 - ▶ 2^{31} n'existe pas, seulement $2^{31}-1$
-



Un autre exemple, en Java

► Recherche par dichotomie dans un tableau trié

```
► public static int binarySearch(int[] a, int key) {  
    int low = 0;  
    int high = a.length - 1;  
  
    while (low <= high) {  
        int mid = (low + high) / 2;  
        int midVal = a[mid];  
  
        if (midVal < key)  
            low = mid + 1  
        else if (midVal > key)  
            high = mid - 1;  
        else  
            return mid; // key found  
    }  
    return -(low + 1); // key not found.  
}
```



Un autre exemple, en Java

▶ Recherche par dichotomie dans un tableau trié

```
▶ public static int binarySearch(int[] a, int key) {  
    int low = 0;  
    int high = a.length - 1;  
  
    while (low <= high) {  
        int mid = (low + high) / 2; // dépassement si low + high > 231-1  
        int midVal = a[mid];  
  
        if (midVal < key)  
            low = mid + 1  
        else if (midVal > key)  
            high = mid - 1;  
        else  
            return mid; // key found  
    }  
    return -(low + 1); // key not found.  
}
```

▶ Solution

```
▶ 6: int mid = low + ((high - low) / 2);
```

▶ Problème

```
▶ Bug présent de le JDK de Sun ! Il a impacté des utilisateurs
```

```
▶ http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html
```



Utile les méthodes formelles ? (1 / 3)

- ▶ Les méthodes d'analyse statique sont-elles vraiment *utile* ?
Juste un *exemple*...
- ▶ <http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>

Paris, 19 July 1996

ARIANE 5

Flight 501 Failure

Report by the Inquiry Board



Utile les méthodes formelles? (2/3)

► Chaîne d'événements techniques

Based on the extensive documentation and data on the Ariane 501 failure made available to the Board, the following chain of events, their inter-relations and causes have been established, starting with the destruction of the launcher and tracing back in time towards the primary cause.



- The launcher started to disintegrate at about $H_0 + 39$ seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher.
- This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine.
- These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the active Inertial Reference System (SRI 2). Part of these data at that time did not contain proper flight data, but showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data.
- The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.

Exception à l'exécution



Utile les méthodes formelles? (3 / 3)

Pourquoi
l'erreur s'est
propagée

► Chaîne d'événements techniques

[...]

Cause
informatique
de l'exception

- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

[...]

Problème de
spécification

- The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time.
- The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.



Recommandations pour Ariane 501

▶ Plus de *tests* !

R2 Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage has to be obtained.

▶ Plus de *formel* (implicite) !

R5 Review all flight software (including embedded software), and in particular :

- Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
- Verify the range of values taken by any internal or communication variables in the software.

▶ ⇒ Utilisation d'*analyse abstraite* (PolySpace, Alain Deutsch)



Analyse abstraite

Analyse abstraite : PolySpace et ASTRÉE

▶ PolySpace

- ▶ Logiciel expérimental puis industrialisé (PolySpace → MathWorks) d'analyse statique de code Ada
- ▶ Vérification **automatique** à **95 %** de l'absence de Run Time Error Pour Ariane 502, 503, ...
 - ▶ Inertial Measurement Unit (30.000 lignes)
 - ▶ Flight software (60.000 lignes)

▶ ASTRÉE

- ▶ Nov. 2003, prouve entièrement **automatiquement** l'**absence de toute erreur à l'exécution** (Run Time Errors) sur le logiciel de contrôle de vol primaire de l'Airbus A340 à commandes de vol électriques
 - ▶ 132.000 lignes de C
 - ▶ 1h20 sur un PC 32 bits 2,8 GHz (300 Mo de mémoire)



Analyse abstraite : interprétation abstraite

- ▶ Basée sur l'*interprétation abstraite*
- ▶ *Approximations* faites sur la sémantique (le sens) du langage C
 - ▶ Formalisé par Patrick et Radhia **Cousot** (fin années 70)
 - ▶ Signale *toutes* les erreurs *possibles* pour des *classes d'erreur*
 - ▶ Division par zéro, débordement de tableau, dépassement de capacité sur des variables entières, ...
 - ▶ Parfois signale des erreurs qui n'en sont pas (*fausses alarmes*)



Analyse abstraite : aperçu

- ▶ Un *exemple* stupide : le signe d'un calcul

- ▶ $\text{Sign}(x) = -1$ si $x < 0$, $+1$ sinon

- ▶ $\text{Sign}(x * y) = \text{Sign}(x) * \text{Sign}(y)$ $\text{Sign}(x / y) = \text{Sign}(x) / \text{Sign}(y)$

- ▶ $\text{Sign}(x + y) = ?$ $\text{Sign}(x - y) = ?$ \Rightarrow *approximations*

- ▶ Analyse abstraite

- ▶ *Aller* de l'espace du programme dans un *espace abstrait*

- ▶ Faire l'*analyse*

- ▶ Déterminer le signe d'une expression, nul ou pas, non dépassement des bornes d'un tableau, etc.

- ▶ Calculer dans un temps *fini* et *raisonnable* des propriétés sur un espace *immense* (potentiellement infini)

- ▶ *Projeter* les résultats de l'analyse dans le programme initial



Analyse abstraite : fondements

- ▶ **Fondement théorique** plus compliqué
 - ▶ Fonctions monotones sur des treillis
 - ▶ Connections de Galois
 - ▶ Opérateurs de *widening* et de *narrowing*
 - ▶ Approximations relationnelles et non relationnelles d'un ensemble de variables par des intervalles, polyèdres, des congruences simples ou linéaires, ...
- ▶ Garantir que l'analyse est **valide**
 - ▶ Tout ce qui est démontré dans l'abstraction l'est aussi sur le vrai système
- ▶ Idéalement, garantir que l'analyse est **complète**
 - ▶ Toute erreur signalée par l'analyse est aussi une erreur sur le vrai système
 - ▶ Tout du moins, **limiter** au maximum les **fausses alarmes**



Analyse abstraite : outils

- ▶ Outils *commerciaux* et *propriétaires*
 - ▶ Astrée : <http://www.absint.de/astree/>
 - ▶ Vérification des logiciels embarqués d'Airbus
 - ▶ Polyspace : <http://www.mathworks.com/products/polyspace>
 - ▶ Issu des vérifications pour Ariane 502
- ▶ Outil *libre* et *gratuit*
 - ▶ Frama-C : <http://frama-c.com>
 - ▶ Framework plus général d'analyse et de preuve sur du code C
 - ▶ Logiciel que nous présenterons



PolySpace de MathWorks

P
r
o
v
e
n

Green:
reliable

Red:
faulty

Gray:
dead

Orange:
unproven

```
static void Pointer_Arithmetic (void)
{
    int array[100];
    int i, *p = array;

    for(i = 0; i < 100; i++, p++)
        *p = 0;

    if(get_bus_status() > 0) {
        if (get_oil_pressure() > 0)
            *p = 5;
        else
            i++;
    }

    i = get_bus_status();
    if (i >= 0) { *(p-i) = 10; }

    if ((0 < i) && (i <= 100)) {
        p = p - i;
        *p = 5;
    }
}
```



Astrée de AbsInt

- ▶ Plaque commerciale proclame **zéro** fausses alarmes !

The screenshot displays the Astrée software interface. The main window shows the analyzed file `(invalid)path/scenarios.c` and the original source `C:\Programme / .../examples/scenarios/src/scenarios.c`. The analyzed code includes:

```
23 | short z;  
24 |  
25 |  
26 |  
27 |  
28 | a = SPEED_SENSOR;  
29 |  
30 |  
31 |  
32 |  
33 | ptr = &ArrayBlock[0];  
34 |  
35 | if (uninitialized_1) {  
36 |   ArrayBlock[15] = 0x15;  
37 | }  
38 |  
39 | if (uninitialized_2) {  
40 |   *(ptr + 15) = 0x10;  
41 | }  
42 |  
43 | z = (short) ((unsigned short)vx +  
44 |             (unsigned short)vy);
```

The original source code shows the corresponding C code with comments:

```
38 | /*  
39 |  * Type cast causing overflow.  
40 |  */  
41 | a = SPEED_SENSOR;  
42 |  
43 | /*  
44 |  * Precise handling of pointer  
45 |  * arithmetic.  
46 |  */  
47 | ptr = &ArrayBlock[0];  
48 |  
49 | if (uninitialized_1) {  
50 |   ArrayBlock[15] = 0x15; // easy case  
51 | }  
52 |  
53 | if (uninitialized_2) {  
54 |   *(ptr + 15) = 0x10; // hard case  
55 | }  
56 |  
57 | /*  
58 |  * Precise handling of compute-through-  
59 |  * overflow arithmetic.  
60 |  * Note that, by default, alarms on  
61 |  * explicit typecasts are  
62 |  * deactivated (see Options->General  
63 |  * ...)
```

The summary panel at the bottom indicates:

- Errors: 2 (2)
- Alarms: 5 (5)
- Warnings: 1
- Coverage: 100%
- Duration: 20s

The summary panel also shows a list of errors and alarms:

- Errors:
 - Definite runtime error during assignment in this context. Analysis stopped for this context.
 - Definite runtime error during assignment in this context. Analysis stopped for this context.
- Alarms:
 - Assertion failure
 - Possible overflow upon dereference
 - Possible overflow upon dereference
 - Out-of-bound array access
 - Overflow in conversion

Analyse abstraite : grille de lecture

- ▶ Domaines d'application / Problèmes possibles
 - ▶ *Code* Ada / C / C++. Vérifier les *fausses alarmes*
- ▶ Niveau d'expertise : *Nul*
- ▶ Niveau d'intervention :
 - ▶ Sur le *code source final*, annotations
- ▶ Couverture du cycle de développement / Fidélité
 - ▶ Appliqué sur le code final, après chaque changement
- ▶ Disponibilité des outils / Niveau d'automatisme
 - ▶ Plusieurs outils disponibles, analyses *automatiques*
- ▶ Expressivité : qu'est-ce que je peux prouver ?
 - ▶ Certaines *classes* de propriétés : non division par zéro, accès hors bornes, dépassement de capacité, ...

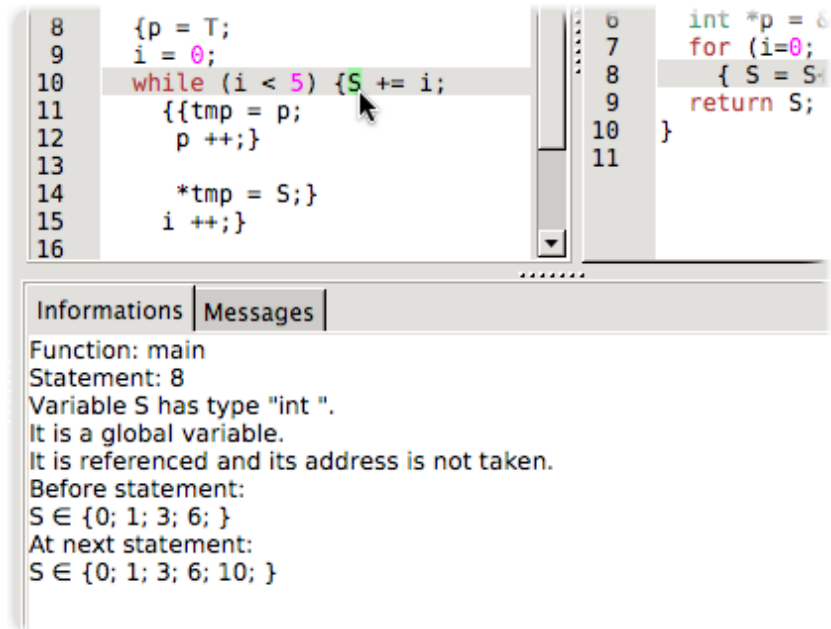


Analyse abstraite de code C avec Frama-C

Un peu de concret !

Qu'est-ce que Frama-C ?

- ▶ Framework d'analyse de **code C** <http://frama-c.com>
 - ▶ Développé par le CEA LIST et l'INRIA-Saclay
 - ▶ Programmé et extensible (*plugins*) en **OCaml**
 - ▶ Différentes **analyses combinables** entre elles
 - ▶ Analyse de valeurs, vérification déductive, *slicing*, code mort, propagation de constantes, dépendances entre variables, ...
 - ▶ Logiciel **libre** et **gratuit**
 - ▶ Liste de diffusion pour le **support**
- ▶ **Interface graphique** pour montrer le résultat des analyses
- ▶ Paquets Debian et Ubuntu
`frama-c`



The screenshot displays the Frama-C graphical interface. The top pane shows C code with a `while` loop. The variable `S` is highlighted in the code. The bottom pane shows analysis results for the selected statement.

```
8   {p = T;
9   i = 0;
10  while (i < 5) {S += i;
11      {{tmp = p;
12         p ++;}}
13
14      *tmp = S;}
15  i ++;}
16
```

```
6   int *p = 0;
7   for (i=0;
8       { S = S;
9         return S;
10      }
11
```

Informations | Messages

Function: main
Statement: 8
Variable S has type "int".
It is a global variable.
It is referenced and its address is not taken.
Before statement:
S ∈ {0; 1; 3; 6; }
At next statement:
S ∈ {0; 1; 3; 6; 10; }

Plugin fondamental : analyse de valeurs

- ▶ Activé par l'option `-val` : calcul la valeur des variables
- ▶ Valeurs pour des variables *entières*
 - ▶ *Énumération* : $a \in \{4; 5; \}$
 - ▶ *Intervalle* : $i \in [0..100]$ $a \in [--..--]$
 - ▶ *Intervalle avec périodicité* : $i \in [2..42]\%2,10$
 - ▶ Toutes les valeurs de l'intervalle ayant pour reste 2 après division par 10 :
2, 12, 22, 32, 42
- ▶ Valeurs pour des *flottants*
 - ▶ Valeur *exacte* (3.0) et *intervalle* ($[-1.0..1.0]$)
- ▶ Valeurs pour des *tableaux* et *pointeurs*
 - ▶ *Ensemble d'adresses* sous la forme *base + offset* (en octets)
 - ▶ $p \in \{ \{ \&a ; \&b ; \} \}$
 - ▶ $p \in \{ \{ \&t + \{0; 4; 8; 12; \} ; \&s + \{ 0; \} ; \} \}$



Exemples analyse de valeurs : démo !

The screenshot displays a code analysis tool interface. The top menu bar includes 'File', 'Project', 'Analyses', and 'Help'. Below the menu is a toolbar with various icons. On the left, a 'Source file' pane shows a project structure for 'value-analysis.c' with sub-items: 'main', 'loop4', 'loop3', 'loop2', 'loop1', 'more_complex', 'simple', and 'g_var'. The main editor area shows the source code for 'value-analysis.c' with the following content:

```
int g_var ;
int simple(int x )
{
    int a ;
    int *p ;
    int __retres ;
    p = & a;
    *p = 3;
    a = 4;
    __retres = *p;
    return (__retres);
}

int more_complex(int x )
{
    int t[10] ;
    int *p ;
    int __retres ;
    p = & t[3];
    t[3] = 2;
}
```

On the right, a separate window titled 'value-analysis.c' shows a zoomed-in view of the code, highlighting lines 9 and 19:

```
3 int simple(int x)
4 {
5     int a, b, *p;
6
7     p = &a;
8
9     *p = 3;
10    a = 4;
11
12    return *p;
13 }
14
15 int more_complex(int x)
16 {
17     int t[10], *p;
18
19     p = &t[3];
20
```

At the bottom, a 'Slicing' panel is visible with options for 'Activate' (set to 'None'), 'Enable' (checkbox), and 'Libraries' (checkbox). Below it, a 'Metrics' section is partially visible. The bottom pane shows analysis results for the 'simple' function, statement 2 (line 9):

Function: simple
Statement: 2 (line 9 in value-analysis.c)
Variable p has type "int *".
It is a local variable.
It is referenced and its address is not taken.
Before statement:
p ∈ {{ &a ;}}
At next statement:
p ∈ {{ &a ;}}

Frama-C pour l'analyse de code

- ▶ `frama-c-gui -val value-analysis.c`
- ▶ Permet de faire des vérifications *exhaustives*
 - ▶ Considère tous les cas possibles
 - ▶ Considère des cas très *difficilement accessibles* au test
 - ▶ Chaque alarme correspond à l'ajout d'une assertion
 - ▶ *Bug* potentiel !
 - ▶ Assertion permet de *continuer l'analyse* (sous hypothèse que l'assertion est vérifiée)
- ▶ Parfois l'analyse donne des sur-approximations
 - ▶ *Fausse alarmes*
 - ▶ Augmenter la finesse de l'analyse : paramètre `-slevel`



Alarmes générées

- ▶ **Division et modulo par zéro**
 - ▶ `10 / y` `10 % y`
- ▶ **Décalage indéfini (hors de [0 . . 31])**
 - ▶ `1 << c`
- ▶ **Dépassement en arithmétique signée**
 - ▶ Avec option `-val-signed-overflow-alarms`
- ▶ **Valeurs flottantes dangereuses**
 - ▶ Quand une opération peut rendre une valeur infinie ou NaN
 - ▶ Utilisation d'une autre valeur (ex. entier) comme flottant
- ▶ **Variables non initialisées et pointeurs sur des variables locales**
- ▶ **Accès mémoire invalides**
 - ▶ Par exemple, dépassement de tableau
- ▶ **Comparaisons de pointeurs ou effets de bord dangereux**



Exemple : valeur absolue, démo (1/2)

The screenshot shows a code editor with the following components:

- Source file list:** Shows 'absolute.c' with sub-files 'main' and 'z_abs_x'.
- Code editor (left):** Contains the implementation of `z_abs_x` and `main`. The assertion `/*@ assert (((-0x7FFFFFFF-1) ≤ -x) ∧ (-x ≤ 2147483647));` is highlighted in green.
- Code editor (right):** Shows the implementation of `absolute.c` with line numbers 1-20. Line 6, `z = -x;`, is highlighted in grey.
- Console (bottom):** Displays an error message: `absolute.c:6 Kernel signed overflow, should be between -2147483648 and 2147483647: assert (((-0x7FFFFFFF-1) ≤ -x) ∧ (-x ≤ 2147483647));`
- Left sidebar:** Contains 'Slicing' and 'Metrics' sections.

Exemple : valeur absolue (2/2)

- ▶ `frama-c-gui -val -val-signed-overflow-alarms absolute.c`
- ▶ L'analyse de valeur de Frama-C
 - ▶ A *inféré* que la variable `x` a une valeur dans le domaine $[-\infty, +\infty]$
 - ▶ En déduit que l'opération « `-x` » est une *erreur* pour $x = -2^{31}$
 - ▶ Poursuit l'analyse en *supposant* que $x \neq -2^{31}$
- ▶ Insertion d'une assertion
 - ▶ À *prouver* par la suite avec d'autres hypothèses
 - ▶ *Ou modifier* le code pour que l'erreur ne se produise pas
 - ▶ Ajouter un test par exemple



Autres exemples

▶ Alarmes

▶ Démo !

▶ `frama-c-gui -val -val-signed-overflow-alarms alarms.c`

▶ Division possible par zéro : utilisation de `-slevel n`

▶ Prendre en compte au plus n chemins différents lors de l'analyse

▶ Démo !

▶ Comparer

▶ `frama-c-gui -val possible-zero.c`

▶ `frama-c-gui -val -slevel 2 possible-zero.c`





Conclusion

Pour finir

- ▶ L'analyse abstraite est un *cadre théorique* d'analyse de code
 - ▶ Garantie formelle que l'analyse *trouvera toutes* les erreurs cherchées
 - ▶ Risque de *fausses alarmes*
- ▶ Théorie complexe mais *outils plus faciles* à prendre en main
 - ▶ Applicable sur du *code réel C ou Ada* (ex. Airbus et Ariane)
 - ▶ De taille industriel même s'il y a des *limites* (50.000 ~ 100.000 lignes)
 - ▶ Plus facile à mettre en œuvre que d'autres outils formels
- ▶ Ne résout pas tout !
 - ▶ Bug dans les outils ? Bug dans les spécifications (cf. ex. Ariane) ? Propriétés compliquées ? Terminaison ?
 - ▶ Mais ce n'est pas une raison pour ne pas l'*utiliser* ! 😊





Backup slides

Pour aller plus loin !

Preuve de code

- ▶ Plugin pour *prouver* des propriétés
 - ▶ Plugin *Jessie* avec outils *Why* + prouveurs (Alt-Ergo, Coq, ...)
 - ▶ Utilisation de la logique de Hoare
 - ▶ Prouver *absence d'erreur à l'exécution* (Run Time Error)
 - ▶ Prouver la *terminaison*
- ▶ Méthode de travail
 1. *Annotation* de code
 2. *Analyse*
 3. Génération d'*obligations de preuves*
 4. *Preuves* (automatiques ou manuelles)
- ▶ Outils plus difficiles à mettre en œuvre



Logique de Hoare : exemple Framac-C

- ▶ Framac-C / Jessie : logique de Hoare sur du vrai code C
- ▶ Utilisation de *démonstrateurs automatiques* (SMT solvers) pour les preuves en utilisant *Why*

- ▶ *Invariant de boucle*

- ▶ Construction *progressive* de la propriété requise

	counters
0	12
1	5
2	15
3	10

j ↑
counters[winner] ≥
counters[j]
↓
 i

$\backslash result = winner = 2$

```
/*@ requires l < num_candidates && num_candidates < MAX_CANDIDATES;
    assigns \nothing;
    ensures \result >= l && \result < num_candidates;
    ensures \forallall integer i;
        l <= i < num_candidates ==> counters[\result] >= counters[i];
*/
int compute_winner(void)
{
    int i, winner;

    winner = 1; /* "No vote" is NOT taken into account */
    /*@ loop invariant 2 <= i && i < MAX_CANDIDATES;
        loop invariant \forallall integer j;
            l <= j < i ==> counters[winner] >= counters[j];
        loop invariant winner >= l && winner < num_candidates;
    */
    for (i = 2; i < num_candidates; i++) {
        if (counters[i] > counters[winner]) { winner = i; }
    }
    return winner;
}
```



Exemple Frama-C : preuves automatiques

gWhy : Easy proof with easy tool

File Configuration Proof

Proof obligations	Alt-Ergo 0.8	Alt-Ergo 0.8 (Select)	Simplify (uninstalled) (Graph)	Yices (uninstalled) (SS)	CVC3 1.5 (SS)	Simplify (uninstalled) (Strat)
Function compute_results Safety						
Function compute_winner Default behavior	✓					
1. initialization of loop invariant	●	—	—	—	—	—
2. initialization of loop invariant	●	—	—	—	—	—
3. initialization of loop invariant	●	—	—	—	—	—
4. initialization of loop invariant	●	—	—	—	—	—
5. initialization of loop invariant	●	—	—	—	—	—
6. preservation of loop invariant	●	—	—	—	—	—
7. preservation of loop invariant	●	—	—	—	—	—
8. preservation of loop invariant	●	—	—	—	—	—
9. preservation of loop invariant	●	—	—	—	—	—
10. preservation of loop invariant	●	—	—	—	—	—
11. preservation of loop invariant	●	—	—	—	—	—
12. preservation of loop invariant	●	—	—	—	—	—
13. preservation of loop invariant	●	—	—	—	—	—
14. preservation of loop invariant	●	—	—	—	—	—
15. preservation of loop invariant	●	—	—	—	—	—
16. postcondition	●	—	—	—	—	—
17. postcondition	●	—	—	—	—	—
18. postcondition	●	—	—	—	—	—
Function compute_winner Safety	✓					
1. pointer dereferencing	●	—	—	—	—	—
2. pointer dereferencing	●	—	—	—	—	—
3. pointer dereferencing	●	—	—	—	—	—
4. pointer dereferencing	●	—	—	—	—	—
5. check arithmetic overflow	●	—	—	—	—	—
6. check arithmetic overflow	●	—	—	—	—	—
7. check arithmetic overflow	●	—	—	—	—	—

```

H8: i_1_0 = result0
i_1_0_0: int32
winner0: int32
H9: true
H10: (2 <= integer_of_int32(i_1_0_0) and integer_of_int32
(i_1_0_0) < 20) and
(forall j_4:int.
  1 <= j_4 and j_4 < integer_of_int32(i_1_0_0) ->
  integer_of_int32(select(int_P_int_M_counters_ll,
  shift(counters, integer_of_int32
(winner0)))) >=
  integer_of_int32(select(int_P_int_M_counters_ll, shift
(counters, j_4)))) and
(integer_of_int32(winner0) >= 1 and
integer_of_int32(winner0) < integer_of_int32
(num_candidates))
H11: integer_of_int32(i_1_0_0) < integer_of_int32
(num_candidates)
H12: offset_min(int_P_counters_ll_alloc_table, counters) <=
integer_of_int32(i_1_0_0) and
integer_of_int32(i_1_0_0) <= offset_max
(int_P_counters_ll_alloc_table,
counters)
result1: int32
H13: result1 = select(int_P_int_M_counters_ll,
  shift(counters, integer_of_int32(i_1_0_0)))
H14: offset_min(int_P_counters_ll_alloc_table, counters) <=
integer_of_int32(winner0) and
integer_of_int32(winner0) <= offset_max
(int_P_counters_ll_alloc_table,
counters)
result2: int32
H15: result2 = select(int_P_int_M_counters_ll,
  shift(counters, integer_of_int32(winner0)))
H16: integer_of_int32(result1) > integer_of_int32(result2)
winner1: int32
H17: winner1 = i_1_0_0

-2147483648 <= integer_of_int32(i_1_0_0) + 1

loop invariant \forall int i;
  1 <= i < num_candidates ==> counters[winner] >=
counters[i];
loop invariant winner >= 1 && winner < num_candidates;
*/
for (i = 2; i < num_candidates; i++) {
  if (counters[i] > counters[winner]) { winner = i; }
}
return winner;

/*@ requires 1 < num_candidates && num_candidates <
MAX_CANDIDATES;
requires \forall int i; 0 <= i < MAX_CANDIDATES
==> counters[i] < MAX_VOTES_PER_CANDIDATE
&& counters[i] >= 0;
assigns \nothing;
*/

```

Timeout: 10 Pretty Printer file: evoting.cVC: check arithmetic overflow

Connexion de Galois (Cousot, 2005)

$$\langle \mathcal{D}, \sqsubseteq \rangle \begin{array}{c} \xleftarrow{\gamma} \\ \xrightarrow{\alpha} \end{array} \langle \overline{\mathcal{D}}, \sqsubseteq \rangle$$

iff $\forall x, y \in \mathcal{D} : x \sqsubseteq y \implies \alpha(x) \sqsubseteq \alpha(y)$

$\wedge \forall \bar{x}, \bar{y} \in \overline{\mathcal{D}} : \bar{x} \sqsubseteq \bar{y} \implies \gamma(\bar{x}) \sqsubseteq \gamma(\bar{y})$

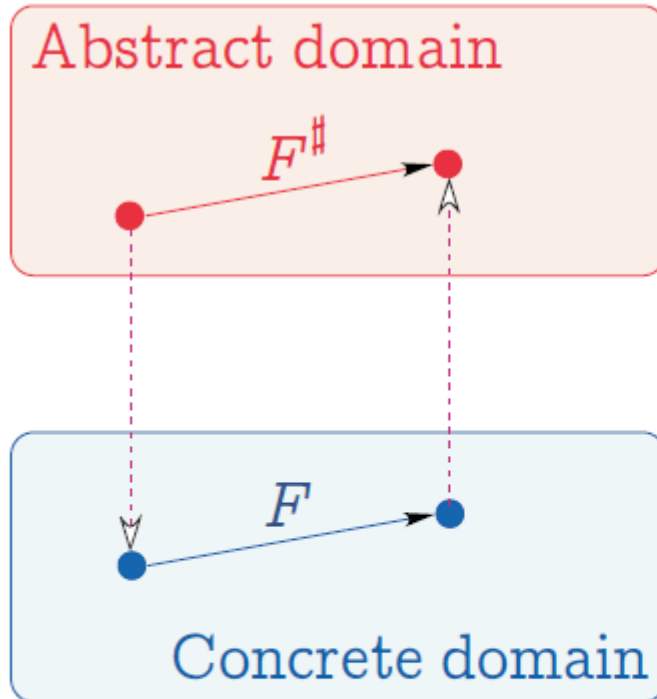
$\wedge \forall x \in \mathcal{D} : x \sqsubseteq \gamma(\alpha(x))$

$\wedge \forall \bar{y} \in \overline{\mathcal{D}} : \alpha(\gamma(\bar{y})) \sqsubseteq \bar{y}$

iff $\forall x \in \mathcal{D}, \bar{y} \in \overline{\mathcal{D}} : \alpha(x) \sqsubseteq \bar{y} \iff x \sqsubseteq \gamma(\bar{y})$



Abstraction de fonction



Function abstraction

$$F^\# = \alpha \circ F \circ \gamma$$

i.e. $F^\# = \rho \circ F$

$$\langle P, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle Q, \sqsubseteq \rangle \Rightarrow$$

$$\langle P \xrightarrow{\text{mon}} P, \dot{\sqsubseteq} \rangle \xleftrightarrow[\lambda F \cdot \alpha \circ F \circ \gamma]{\lambda F^\# \cdot \gamma \circ F^\# \circ \alpha} \langle Q \xrightarrow{\text{mon}} Q, \dot{\sqsubseteq} \rangle$$



Licence de cette présentation

- ▶ Cette présentation est sous licence Art Libre 1.3
 - ▶ <http://artlibre.org/licence/lal>
 - ▶ Copyright 2011 David MENTRÉ
 - ▶ Exception : images crash Ariane et connexion de Galois, propriétés de leur auteur

