Estimation de WCET haut-niveau avec Interprétation abstraite et Programmation par contraintes

Aurèle Barrière

17 mai - 22 juillet 2016

Équipe : CELTIQUE

Estimation WCET

WCET : *worst case execution time*

On veut une majoration **sûre** du WCET.

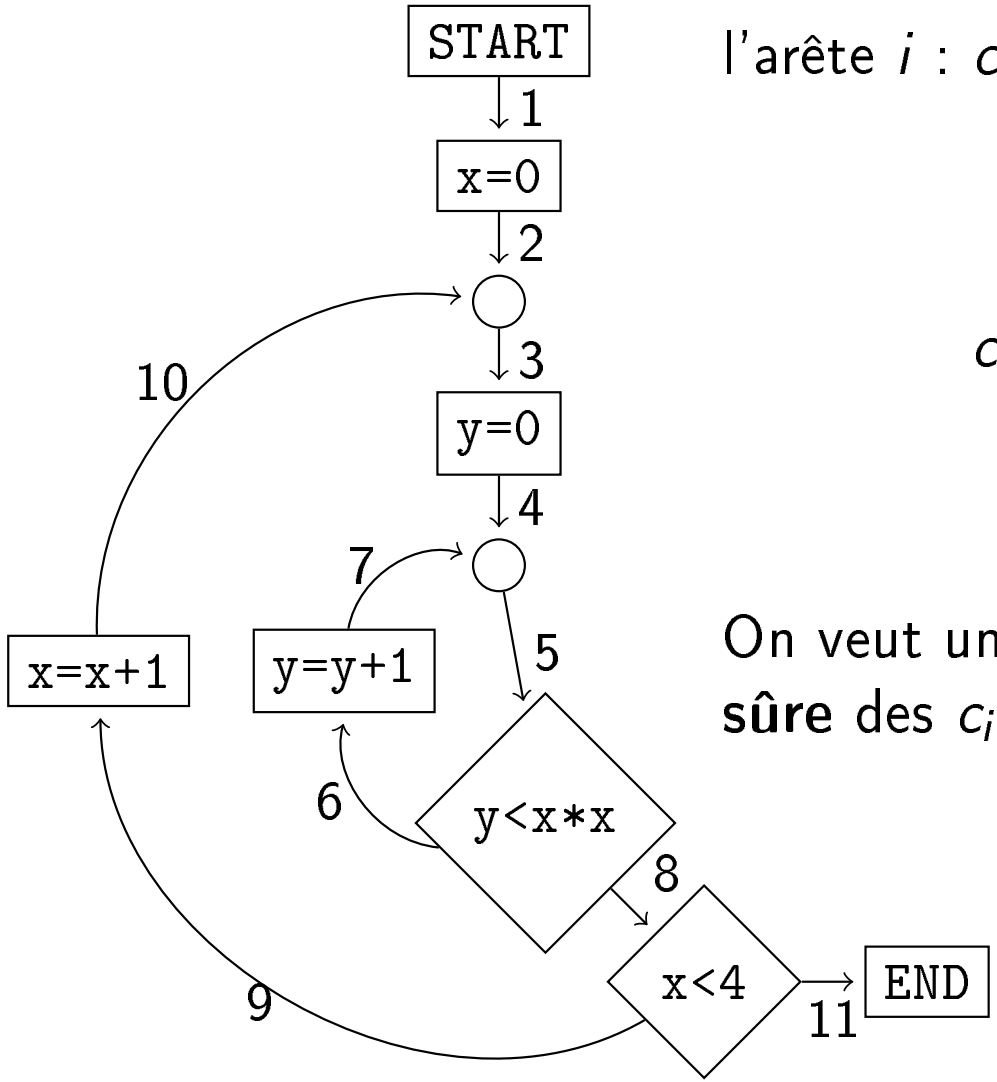
Analyse haut-niveau ou analyse de flot du programme. Estimer le nombre d'exécutions de chaque instruction.

Analyse bas-niveau. Estimer le temps d'exécution de chaque instruction dans le pire cas.

Analyse des anomalies. Prendre en compte les phénomènes de *caches*, *pipelines* et leur influence sur le WCET.

[Wilhelm et al., 2008]

CFG et notations



Nombre d'exécutions de l'arête i : c_i.

$c_3 = 5$

On veut une majoration **sûre** des c_i.

Hypothèses

On ne considérera que des programmes qui vérifient les hypothèses suivantes :

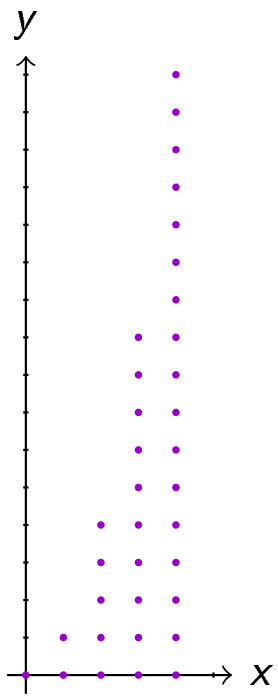
- Les programmes sont déterministes.
- Les programmes terminent.

Sous ces hypothèses, un programme ne passe pas deux fois dans la même arête avec le même état mémoire [Bygde et al., 2011].

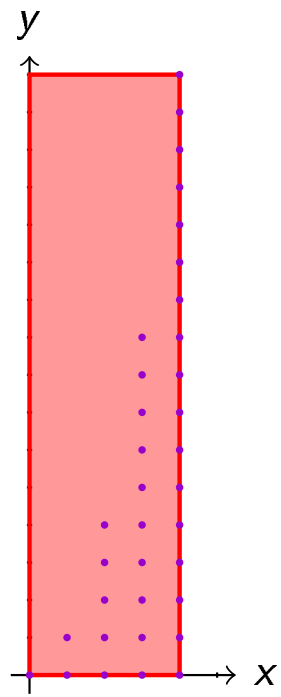
Interprétation abstraite

Sur-approximation de la sémantique d'un programme [Cousot and Cousot, 1977].

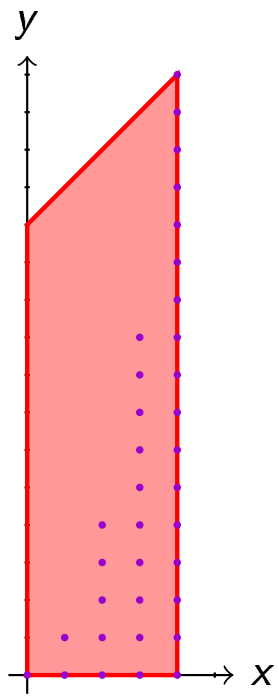
À chaque arête i, on associe un ensemble d'états de la mémoire. Tout état de la mémoire accessible appartient à cet ensemble.



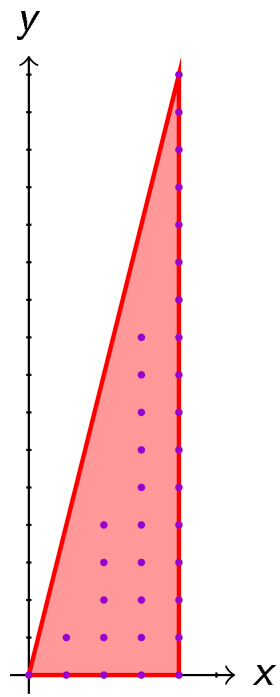
États atteignables à l'arête 5.



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



Octogones
$\pm x \pm y \leq c$



Polyèdres
Toute contrainte linéaire

Programmation par contraintes

Contrainte

Une relation logique du premier ordre sur un langage de contraintes [Rossi et al., 2006].

CSP : *constraint satisfaction problem*

X ensemble de variables

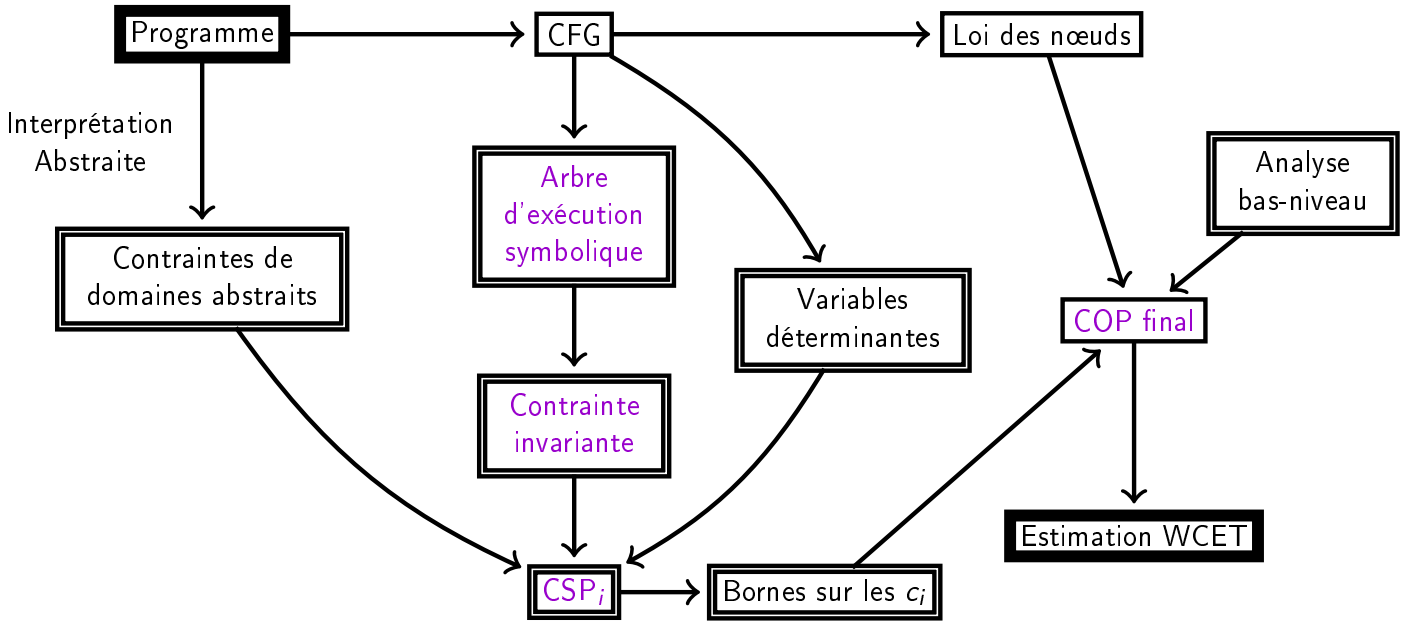
D ensemble de domaines pour chaque variable

C ensemble de contraintes sur les variables

Exemple

$X = \{x_1, x_2\}$, $D = \{[\![0, 2]\!], [\![0, 4]\!]\}$,
$C = \{x_1 = 0, \quad x_1^2 + x_2 \geq 7, \quad x_1 \ mod \ 2 = 0\}$ est un CSP sans solution. Par contre, si $D = \{[\![0, 2]\!], [\![0, 7]\!]\}$, alors on a la solution $x_1 = 0, x_2 = 7$.

Contribution



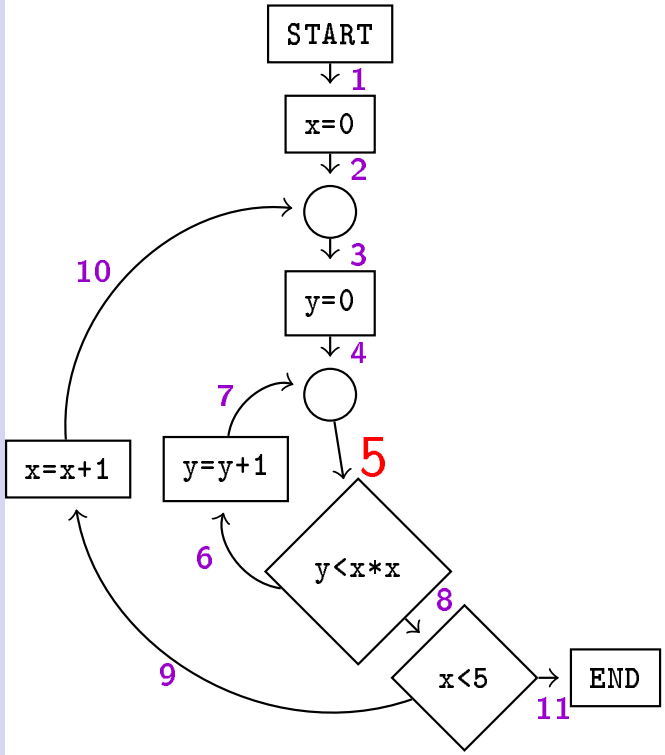
CSP$_i$

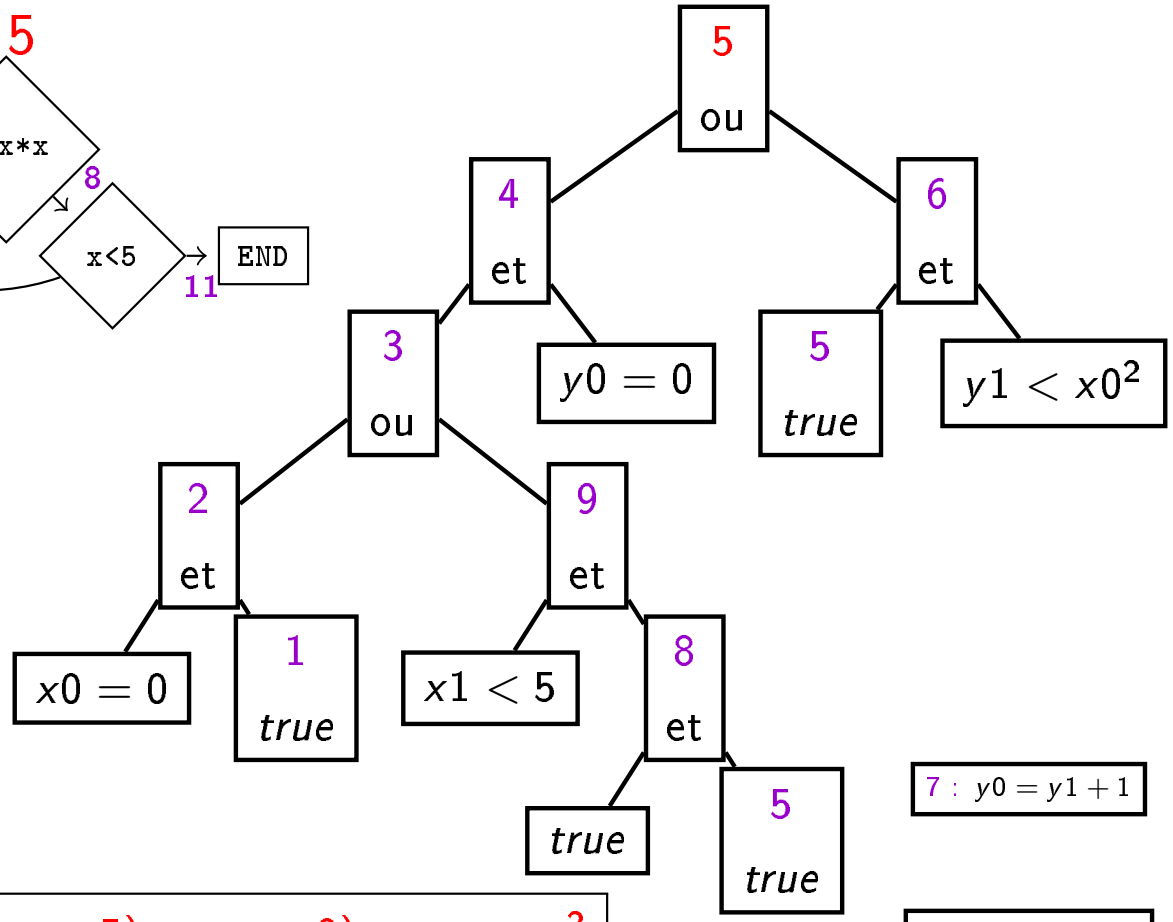
Un CSP dont chaque état mémoire atteignable à l'arête i est une solution.

Exemple

$$X = \{x, y\}$$
$$D = \mathbb{Z} \times \mathbb{Z}$$
$$C = \{x \geq 0, x \leq 4, y \geq 0, y \leq 16, y \leq x^2\}$$

Arbre d'exécution symbolique





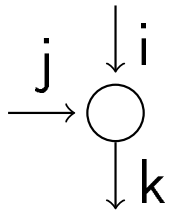
$((x_0 = 0 \text{ ou } x_1 < 5) \text{ et } y_0 = 0) \text{ ou } y_1 < x_0^2$

7 : $y0 = y1 + 1$

10 : $x0 = x1 + 1$

COP final

Pour chaque c_i, on a trouvé en comptant les solutions du CSP_i une borne b_i telle que $c_i \leq b_i$.
K : contraintes de la loi des nœuds sur le CFG.



$$c_i + c_j = c_k$$

COP final

$$X = \{c_i\} \qquad C = (\bigwedge_i c_i \leq b_i) \bigwedge K$$
$$D_i = \mathbb{N} \qquad O = \sum_i c_i \times v_i$$

Les v_i sont obtenues par analyse bas-niveau.

Résultat final

Majoration sûre de WCET

Implémentation en OCaml

AbSolute

Solveur de contraintes. Peut compter les solutions, travailler avec des variables entières, minimiser une fonction. Utilise déjà l'interprétation abstraite [Pelleau et al., 2013].

SawjaCard

Analyseur statique de JavaCard (Java pour cartes électroniques). Peut faire de l'interprétation abstraite avec le domaine des intervalles. Utilise une autre structure de CFG [Besson et al., 2014].

Implémentation modulaire

Les modules qui, à partir d'un CFG, calculent les CSP$_i$ peuvent être utilisés avec d'autres solveurs de contraintes (CHOCO), d'autres analyseurs statiques (Interproc).

Exemple

Double boucle issue du Benchmark Mälardalen

```
$sh estimate.sh
Abstract interpretation
creating the AI constraints
creating all CSP files
wrote all .csp files
Nodes law written
created edgetime file
Solving CSP c1.csp
Solving CSP c10.csp
Solving CSP c11.csp
Solving CSP c2.csp
Solving CSP c3.csp
Solving CSP c4.csp
Solving CSP c5.csp
Solving CSP c6.csp
Solving CSP c7.csp
Solving CSP c8.csp
Solving CSP c9.csp
init{
int   c1   = [0; 14641 ];
int   c10  = [0; 75 ];
int   c11  = [0; 1 ];
int   c2   = [0; 121 ];
int   c3   = [0; 194 ];
int   c4   = [0; 75 ];
int   c5   = [0; 1950 ];
int   c6   = [0; 1875 ];
int   c7   = [0; 1875 ];
int   c8   = [0; 75 ];
int   c9   = [0; 75 ];
}
```

```
objective{
-(0
+ c1 * 0 + c10 * 1
+ c11 * 1 + c2 * 1
+ c3 * 0 + c4 * 1
+ c5 * 0 + c6 * 1
+ c7 * 1 + c8 * 1
+ c9 * 1
)
}

constraints{

//Kirchhoff's law
c1 = 1;
c1 = c2;
c2 + c10 = c3;
c3 = c4;
c4 + c7 = c5;
c5 = c6 + c8;
c6 = c7;
c8 = c9 + c11;
c9 = c10;
c11 = 1;
}

solving ends
Unique solution
best value:-4050.000000
sure:c1%:[1.;1.] c10%:[74.;74.]
...
```

Résultats

Mälardalen WCET Benchmark en C

`http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`

Programme	IA et PPC	IA seule Intervalles	IA seule Polyèdres
Boucle `for` simple	✓	✓	✓
Boucles `for` imbriquées	✓	✓	✓
Boucles `for` triangulaires	✓		✓
Test de primalité	✓		

IA : Interprétation abstraite
PPC : Programmation par contraintes
✓ : estimation exacte des c_i

Conclusion

Améliorations

On tire profit de l'expressivité des langages de contraintes et de la précision de la programmation par contraintes pour affiner l'approximation faite par l'interprétation abstraite.

Dès que le programme exhibe dans son code des contraintes (même non linéaires), on peut les récupérer.

Continuation

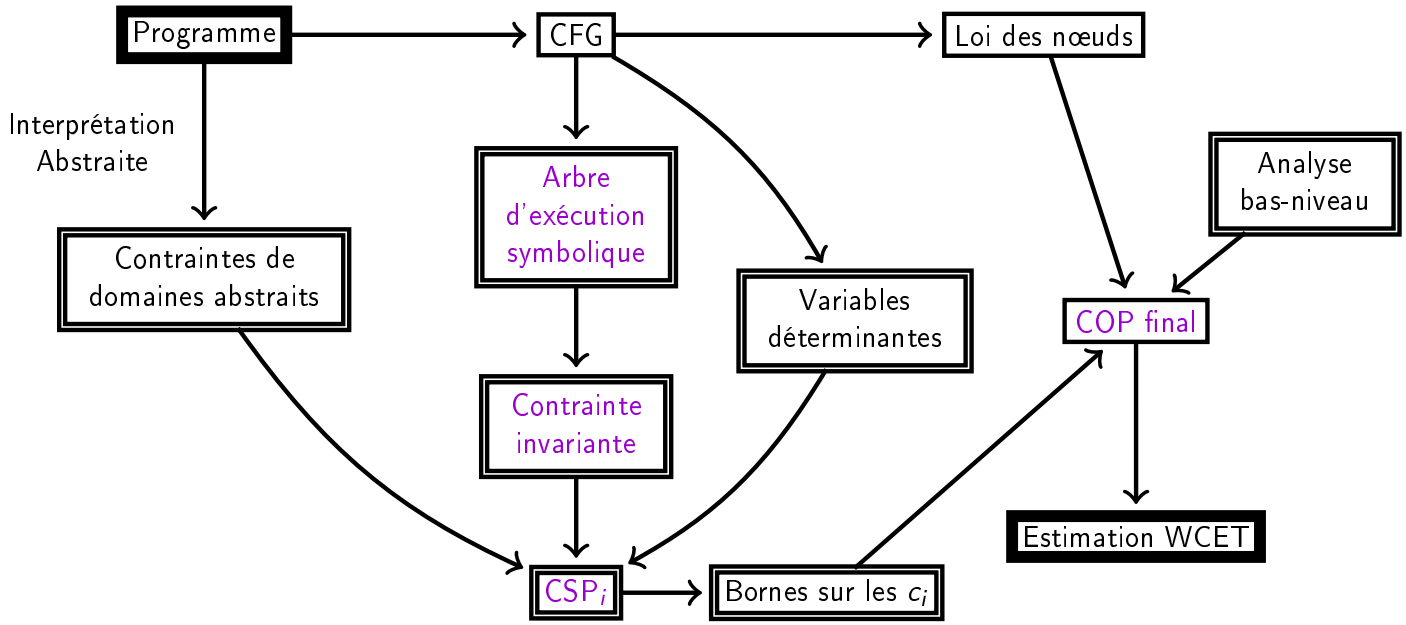
Intégrer l'estimation à SawjaCard.

Heuristiques de recherche.

Agrandir le langage étudié.

Expression paramétrique du WCET.

Résumé de la méthode

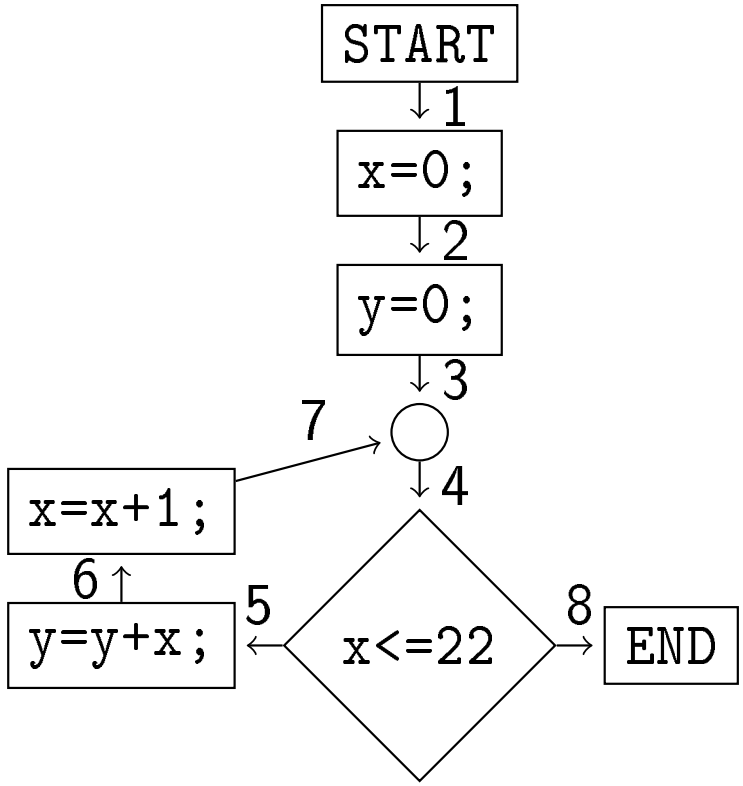


Implémentation et rapport

`https://github.com/Aurele-Barriere/WCET_Estimation`

Variables déterminantes

En 6, $x \in [0, 22]$, $y \in [0, 253]$. 5566 solutions.
Pourtant, $c_6 = 23$. Une exécution par valeur de x.



Compter le nombre de solutions qui ne diffèrent que sur un sous-ensemble des variables.

Bibliographie I

http://www.astree.ens.fr.

http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

https://github.com/Aurele-Barriere/WCET_Estimation.

Besson, F., Jensen, T., and Vittet, P. (2014).
SawjaCard : A Static Analysis Tool for Certifying Java Card Applications.
In 21st International Static Analysis Symposium (SAS 2014), volume 8858, pages 51 – 67, Munich, Germany. Springer.

Bygde, S. (2009).
Static WCET Analysis based on Abstract Interpretation and Counting of Elements.
PhD thesis, School of Innovation, Design and Engineering, Malardalen University.

Bygde, S., Ermedahl, A., and Lisper, B. (2011).
An efficient algorithm for parametric WCET calculation.
Journal of Systems Architecture - Embedded Systems Design, 57(6) :614–624.

Cousot, P. and Cousot, R. (1977).
Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '77, pages 238–252, New York, NY, USA. ACM.

Bibliographie II

Cousot, P. and Cousot, R. (2014).

Abstract interpretation : past, present and future.

In Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014, pages 2 :1–2 :10.

Cousot, P. and Halbwachs, N. (1978).

Automatic discovery of linear restraints among variables of a program.

In Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '78, pages 84–96, New York, NY, USA. ACM.

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., and Zadeck, F. K. (1991).

Efficiently computing static single assignment form and the control dependence graph.

ACM Trans. Program. Lang. Syst., 13(4) :451–490.

Gawlitza, T. M. and Monniaux, D. (2012).

Invariant generation through strategy iteration in succinctly represented control flow graphs.

Logical Methods in Computer Science, 8(3).

Li, Y.-T. S. and Malik, S. (1995).

Performance analysis of embedded software using implicit path enumeration.

In Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference, DAC '95, pages 456–461, New York, NY, USA. ACM.

Bibliographie III

Miné, A. (2004).
Domaines numériques abstraits faiblement relationnels.
PhD thesis, Ecole Polytechnique.

Pelleau, M. (2012).
Domaines abstraits en programmation par contraintes.
PhD thesis, Université de Nantes.

Pelleau, M., Miné, A., Truchet, C., and Benhamou, F. (2013).
A constraint solver based on abstract domains.
In Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings, pages 434–454.

Rossi, F., van Beek, P., and Walsh, T., editors (2006).
Handbook of Constraint Programming, volume 2 of Foundations of Artificial Intelligence.
Elsevier.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenström, P. (2008).
The worst-case execution-time problem : Overview of methods and survey of tools.
ACM Trans. Embed. Comput. Syst., 7(3) :36 :1–36 :53.