# Calcul de limites de boucles sur des programmes binaires

## Master 2 Recherche Informatique et Télécommunications

Florian Birée
Hugues Cassé (encadrant)

Équipe TRACES
IRIT
Université Paul Sabatier

27 juin 2012

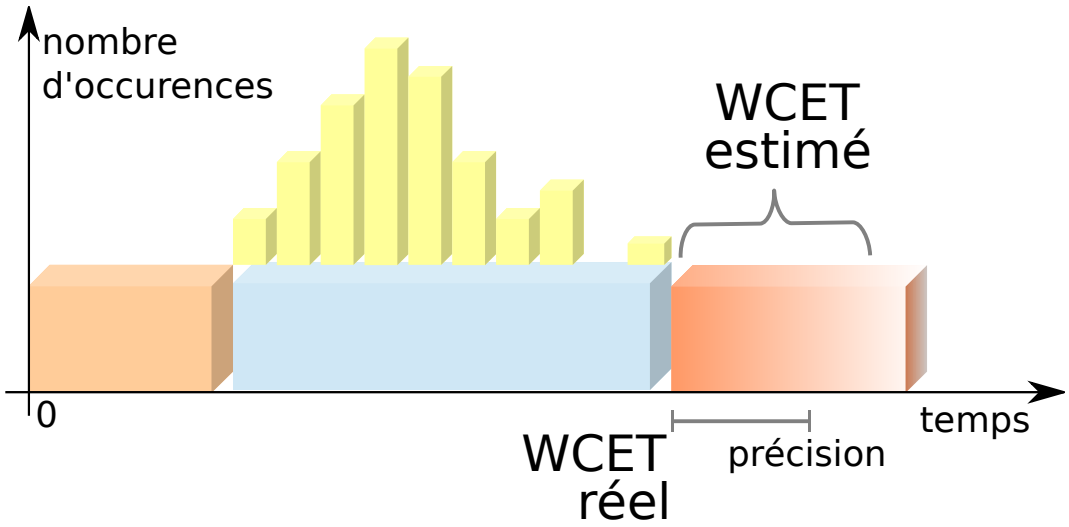


# Plan

# Le calcul de WCET par analyse statique

## WCET : *Worst Case Execution Time*

Domaine : systèmes *critiques temps-réel*.




Calcul du WCET par analyse statique (analyse du matériel et du programme).

# Le calcul de limites de boucles

## Sur les binaires !

Généralement : analyse sur le code source (oRange, TuBound, etc)

- code plus expressif ;
- transformations plus faciles ;
- plus proche de l'utilisateur.

Problèmes :

- lien avec le binaire ;
- les optimisations du compilateur peuvent changer la structure des boucles ;
- disponibilité du code source (fonctions d'émulation, COTS).

Alors, calculons-les directement sur les binaires !

# Exemple de programme

## Du code source au CFG du binaire grâce à Otawa

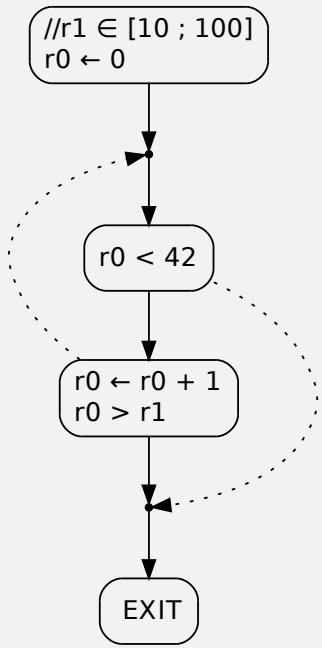
### Code source (C)

```
int k;
/* k ∈ [10 ; 100] */
int i = 0;
while(i < 42)
{
    i++;
    if(i > k)
        break;
}
```

### Assembleur PowerPC

```
; r1 ∈ [10 ; 100]
li      r0,0
b       <main+0x30>
addi    r0,r0,1
cmpwi   cr7,r0,r1
bgt-    cr7,<main+0x3c>
cmpwi   cr7,r0,41
ble+    cr7,<main+0x18>
```

### Graphe de flot de contrôle (CFG)



# Plan

# La forme SSA

## Static Single Assignement

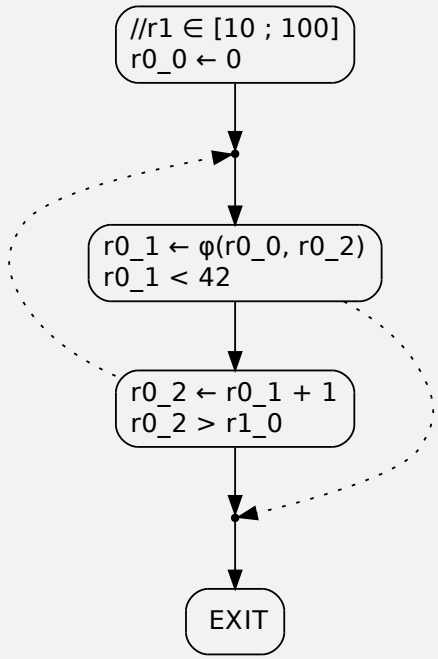
Création de nouvelles variables indicées, une seule affectation par variable, opérateur $\phi$ pour les jointures :

$i = 0$ ;
**if** (...) **then**
  $i = i + a$ ;
**else**
  $i = i + b$ ;
**end if**

$i_0 \leftarrow 0$ ;
**if** (...) **then**
  $i_1 = i_0 + a_0$ ;
**else**
  $i_2 = i_0 + b_0$ ;
**end if**
$i_3 = \phi(i_1, i_2)$ ;

Utilisées pour résoudre les alias entre registres et zones mémoire.

## CFG (SSA)



# Retrouver les expressions d'induction

## Deux sémantiques pour $\phi$

- fusion d'informations incertaines : $\phi$ classique, commutatif.
- $\phi$ des expressions d'induction, récursif, que nous nommerons $\phi^*$.

Les expressions d'induction d'une boucle sont sous la forme d'une expression $\phi^*$ :
$v \leftarrow \phi^*(expr_{init}, expr_{body}(v))$

## Boucle sous la forme $\phi^*$

$$\begin{cases} r0_1 & \leftarrow \phi(r0_0, r0_2) \\ r0_2 & \leftarrow r0_1 + 1 \end{cases}$$

$$\Leftrightarrow r0_1 \leftarrow \phi^*(r0_0, r0_1 + 1)$$

$$expr_{init} = r0_0$$

$$expr_{body}(r0_1) = r0_1 + 1$$

# Construction d'hypothèses de fonctions d'induction

## Plusieurs hypothèses

Fonction d'induction $f$ :
$v \leftarrow f(w), w \in [0; MAX]$
$w$ représente l'itération courante de la boucle.

Hypothèse arithmétique :
$f_a(w) = expr_{init} +$
$(expr_{body}(expr_{init}) - expr_{init})w$

Hypothèse géométrique :
$f_g(w) = expr_{init} \times \left(\frac{expr_{\mathbf{body}}(expr_{\mathbf{init}})}{expr_{\mathbf{init}}}\right)^w$

## Boucle sous la forme $\phi^*$

$$f_a(w) = r0_0 + ((r0_0 + 1) - r0_0)w$$
$$= r0_0 + 1 \times w$$
$$f_g(w) = \ldots$$

# Validation d'une hypothèse

## Propriété de point fixe

$\forall w \in \mathbb{N}$,

$$expr_{body}(f(w)) = f(w+1)$$

Si une hypothèse valide cette propriété, alors elle remplace l'expression $\phi^*$. Sinon, cette dernière est remplacée par $\top$.

## Validation de $f_a$

$$\begin{aligned} expr_{body}(f_a(w)) &= f_a(w+1) \\ \Leftrightarrow f_a(w) + 1 &= f_a(w+1) \\ \Leftrightarrow (r0_0 + w) + 1 &= (r0_0 + w + 1) \\ \Leftrightarrow \text{vrai} & \end{aligned}$$

(faux pour $f_g$)

Expression d'induction de la boucle : $r0_1 \leftarrow r0_0 + 1 \times w$

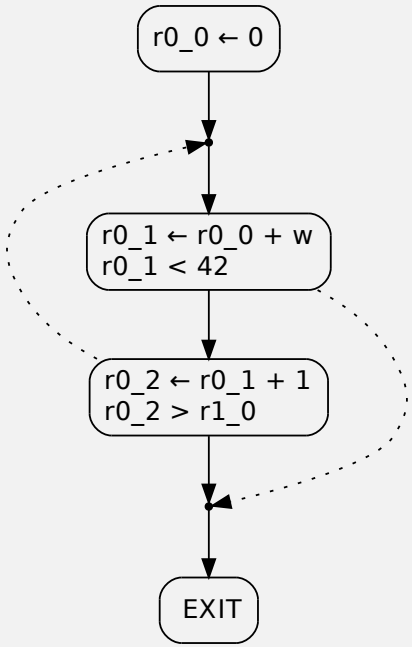
# Plan

# Construction de contraintes sur les arc

## Grammaires des contraintes

$$
\begin{aligned}
\texttt{constr} :: &= \texttt{T} \\
&\mid \texttt{F} \\
&\mid \tau \\
&\mid \texttt{comp} \\
&\mid \neg\ \texttt{constr} \\
&\mid \texttt{constr} \wedge \texttt{constr} \\
&\mid \texttt{constr} \vee \texttt{constr} \\
\\
\texttt{comp} :: &= \texttt{expr} < \texttt{expr} \\
&\mid \texttt{expr} \leq \texttt{expr} \\
&\mid \texttt{expr} > \texttt{expr} \\
&\mid \texttt{expr} \geq \texttt{expr} \\
&\mid \texttt{expr} = \texttt{expr} \\
&\mid \texttt{expr} \neq \texttt{expr}
\end{aligned}
$$

## CFG avec expression d'induction



# Propagation des contraintes sur les arc

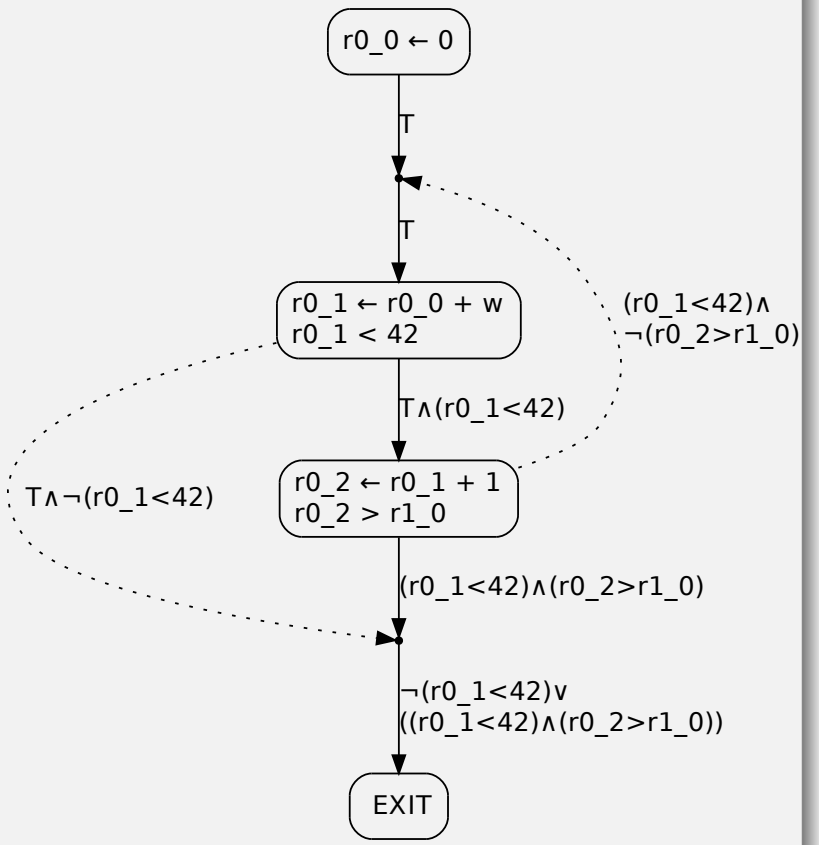
## Interprétation abstraite

Treillis : $\mathtt{F} \sqsubseteq c \sqsubseteq \mathtt{T}$

$$\mathtt{update}((b_1, b_2), constr_{in}) = constr_{in} \wedge constr(b_1, b_2)$$

$$\mathtt{join}(constr_1, constr_2) = constr_1 \vee constr_2$$

## CFG avec expression d'induction



# Plan

# Contrainte de sortie et mise en contexte

## Contrainte de sortie (CS) d'une boucle

Jointure ($\vee$) des contraintes de tous les arcs de sortie d'une boucle. Donne les contraintes sur $w$ en sortie de la boucle.

$\neg CS$ donne les contraintes sur $w$ à l'intérieur de la boucle, et permet de borner $w$.

## Mise en contexte

Opération consistant à remplacer les variables dans CS par leurs expressions, puis par des valeurs issues de différentes sources.

## Contrainte de sortie pour l'exemple

CS : Sortie du `while` $\vee$ sortie du `break`
$\Leftrightarrow \neg(r0_1 < 42) \vee ((r0_1 < 42) \wedge (r0_2 > r1_0))$

Mise en contexte :
$\Leftrightarrow \neg(0 + w < 42) \vee ((0 + w < 42) \wedge (0 + w + 1 > [10; 100]))$

Simplification :
$\Leftrightarrow \neg(w < 42) \vee ((w < 42) \wedge (w > [9; 99]))$

# Intervalle flou des itérations d'une boucle

## Intervalle des itérations

Objectif : construire l'intervalle des valeurs de $w$ pour une boucle (donc des itérations).

## Intervalles flous

$$\begin{pmatrix} N : [l_n; u_s] \\ S : [l_s; u_s] \end{pmatrix}$$

Noyau : ensemble des valeurs certaines.
Support : ensemble des valeurs possibles.

## Conditions de sortie vers intervalle flou

$CS : \neg(w < 42) \vee ((w < 42) \wedge (w > [9; 99]))$

$$w < 42 \mapsto [0; 41]$$

$$w > [9; 99] \mapsto \begin{pmatrix} N : [100; +\infty[ \\ S : [10; +\infty[ \end{pmatrix}$$

# Intervalle flou des itérations d'une boucle

## Transformation de CS en intervalle flou des valeurs de $w$

Utilisation d'opérations ensemblistes pour évaluer les opérations logiques :

$$
\begin{array}{lllll}
CS : & \neg(w < 42) & \vee & ((w < 42) & \wedge \quad (w > [9;99]) \\
 & \complement_{f\mathbb{N}}[0;41] & \cup & ([0;41] & \cap \quad \begin{pmatrix} N : [100;+\infty[ \\ S : [10;+\infty[ \end{pmatrix}) \\
\Leftrightarrow & [42;+\infty[ & \cup & \begin{pmatrix} N : \emptyset \\ S : [11;41] \end{pmatrix} & \\
\Leftrightarrow & & & \begin{pmatrix} N : [42;+\infty[ \\ S : [11;+\infty[ \end{pmatrix} &
\end{array}
$$

$$
\neg CS : \quad \complement_{f\mathbb{N}} \begin{pmatrix} N : [42;+\infty[ \\ S : [11;+\infty[ \end{pmatrix} \Leftrightarrow \begin{pmatrix} N : [0;10] \\ S : [0;41] \end{pmatrix}
$$

# Calcul des limites

## Propriétés d'une boucle

$MIN(b) = \texttt{card}(\mathrm{Noy}(\neg CS))$
$MAX(b) = \texttt{card}(\mathrm{Supp}(\neg CS))$
$TOTAL$ : dépend des boucles englobantes ;
Indications d'exactitude des calculs.

## Propriétés d'un arc

Pour un arc $a$ de contrainte $c$ :
$MIN(a) = \texttt{card}(\mathrm{Noy}(c))$
$MAX(a) = \texttt{card}(\mathrm{Supp}(c))$

## Limites pour l'exemple

$$\neg CS : \begin{pmatrix} N : [0;10] \\ S : [0;41] \end{pmatrix}$$

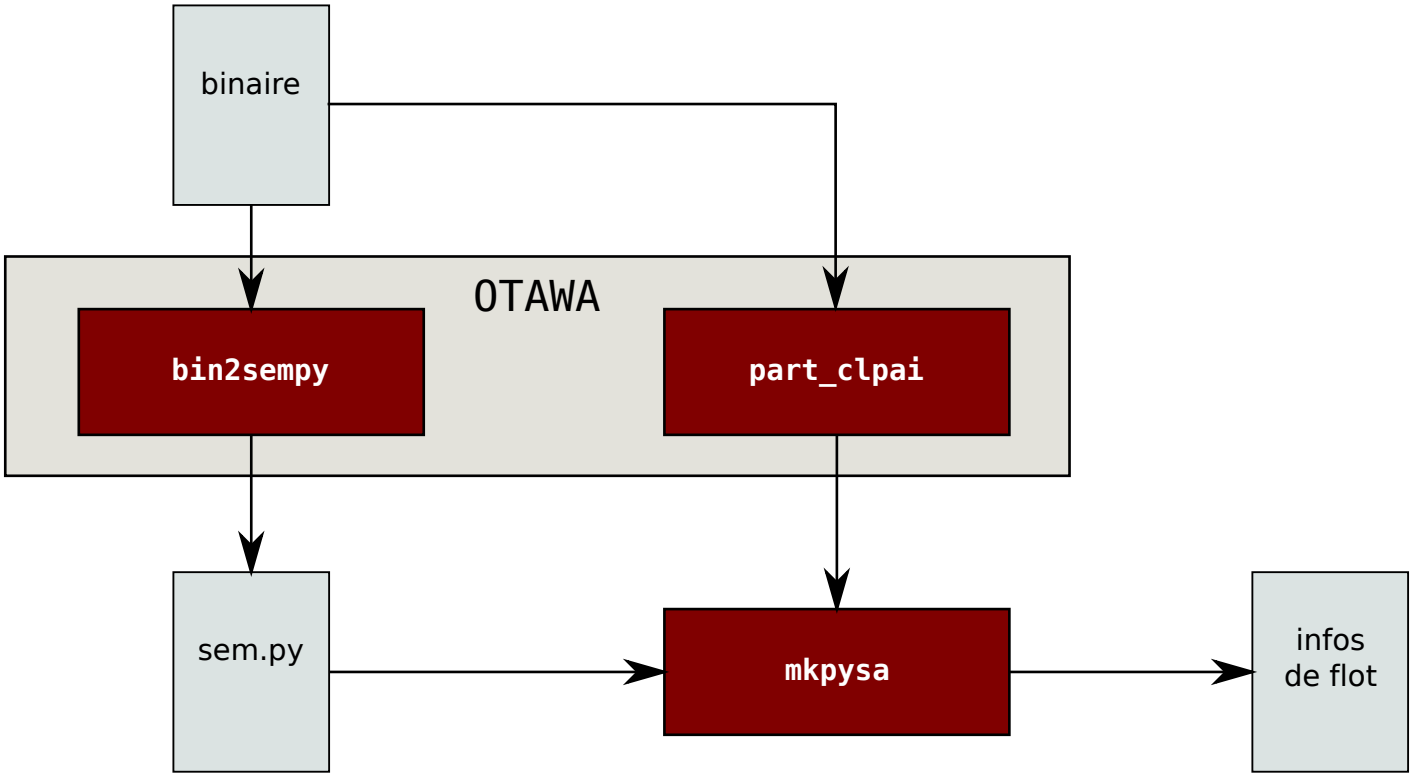
$$MIN(b) = \texttt{card}([0;10]) = 11$$

$$MAX(b) = \texttt{card}([0;41]) = 42$$

# Plan

# Le prototype Pysa

Prototype en Python basé sur le *framework* Otawa :



# Résultats de l'expérimentation du prototype

- jeu de test de 23 boucles ;
- compilées en PowerPC ;
- machine de test : architecture x86, dual-core à 2GHz, 2Go de mémoire vive, Debian GNU/Linux 64bits.
- temps d'analyse : environ 300 instructions à la seconde ;
- sur les boucles analysables par le prototype, résultat au moins aussi précis qu'oRange.

# Plan

# Conclusion

Perspectives d'amélioration :

- support d'autres types d'expressions d'induction (bit à bit, etc) ;
- améliorer la précision des contraintes $\tau$ ;
- ...

Perspectives d'utilisation :

- compléter les outils de calcul de limites de boucle ;
- usage du *MIN* dans le calcul de WCET et de BCET ;
- comportement des caches ;
- analyse des chemins dynamiquement morts ;
- ...

Des questions ?