Minigloca

Vladislas de Haldat

3 juin 2023

Index

Expressions arithmétique et booléenne

$$
\begin{aligned}
a ::= \; & n & n \in \mathbb{Z} \\
\mid \; & x & x \in \mathbb{V} \\
\mid \; & op_A(a_1, a_2) & op_A \in \{+, -, \times\}
\end{aligned}
$$

$$
\begin{aligned}
b ::= \; & \mathbf{true} \mid \mathbf{false} & \\
\mid \; & op_R(a_1, a_2) & op_R \in \{<, =\} \\
\mid \; & op_B(b_1, b_2) & op_B \in \{\wedge, \vee\} \\
\mid \; & \neg b &
\end{aligned}
$$

Déclarations

$$
\begin{aligned}
Stm ::= &\ x := a \\
&\mid s_1 ; s_2 \\
&\mid skip \\
&\mid \text{if } b \text{ then } s_1 \text{ else } s_2 \\
&\mid \text{while } b \text{ do } s \\
&\mid \text{return } a
\end{aligned}
$$

Exemple

```
a := 1;
b := 20;
if a = 3 then
  c := 4
else
  c := 6
endif;
while b < 100 do
  a := b + 1
done;
return c
```

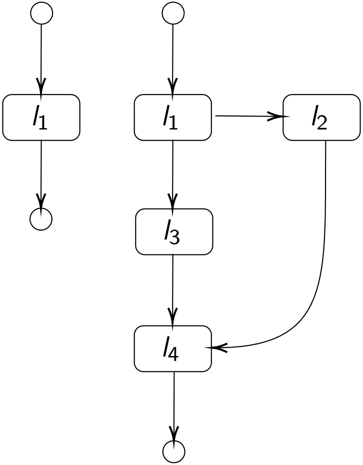
Blocs

$$
\begin{aligned}
Block ::= \; & x := a \\
\mid \; & b \\
\mid \; & skip \\
\mid \; & \text{return } a
\end{aligned}
$$

Notation

Si $s \in Stm$ et $l = \lambda(s)$ alors on notera $s^l \in Stm$ la déclaration munie d'une étiquette.

Graphe de flot de contrôle



Description ensembliste

Définition

Soient les applications

$$vars_a : a \longrightarrow \mathcal{P}(\mathbb{V})$$

$$vars_b : b \longrightarrow \mathcal{P}(\mathbb{V})$$

Les ensembles de variables présentes dans les expressions arithmétique et booléenne.

Description ensembliste

Définition

Soit l'application

$$\begin{aligned} gen : Block &\longrightarrow \mathcal{P}(\mathbb{V}) \\ x := a &\longmapsto vars_a(a) \\ skip &\longmapsto \emptyset \\ b &\longmapsto vars_b(b) \\ \text{return } a &\longmapsto vars_a(a) \end{aligned}$$

Description ensembliste

Définition

Soit l'application

$$\begin{aligned} kill : Block &\longrightarrow \mathcal{P}(\mathbb{V}) \\ x := a &\longmapsto \{x\} \\ skip &\longmapsto \emptyset \\ b &\longmapsto \emptyset \\ \text{return } a &\longmapsto \emptyset \end{aligned}$$

Description ensembliste

Définition

Soient les ensembles

$$
\begin{aligned}
LIVE_{in}[l] &= gen[l] \cup (LIVE_{out}[l] - kill[l]),\\
LIVE_{out}[l] &= \begin{cases} \emptyset & \text{si } succ(l) = \emptyset\,, \\ \displaystyle\bigcup_{p \in succ(s)} LIVE_{in}[p] & \text{sinon}\,. \end{cases}
\end{aligned}
$$

Point fixe

Théorème (Kleene)

Soit $(L, \sqsubseteq)$ un ordre partiellement ordonné, avec un plus petit élément $\perp$ et soit une application $f : L \longrightarrow L$ monotone. Alors il existe un point fixe minimal qui est le suprémum de la suite,

$$\perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp) \sqsubseteq \cdots \sqsubseteq f^k(\perp) \sqsubseteq \cdots$$

Algorithme

Soit s la déclaration
$\mathcal{B} \leftarrow blocks(s)$
Soit $\mathbb{L}$ l'ensemble des étiquettes de $\mathcal{B}$
for $l \in \mathbb{L}$ **do**
  $live_{in}[l] \leftarrow \emptyset$
  $live_{out}[l] \leftarrow \emptyset$
end for
while $live_{in} \neq live'_{in}$ **ou** $live_{out} \neq live'_{out}$ **do**
  for $l \in L$ **do**
    $live_{out}[l] \leftarrow \bigcup_{p \in succ(l)} live_{in}[p]$
    $live_{in}[l] \leftarrow gen[l] \cup (live_{out}[l] - kill[l])$
  end for
end while

Exemple

```
a := 0;                   // empty       {a}
b := a;                   // {a}         {a, b}
while a < 100 do          // {a, b}      {a, b}
  if a = 2 then           // {a, b}      {a, b}
    c := a                // {a, b}      {b, c}
  else
    c := 2 * a;           // {a, b}      {b, c}
    d := b                // {b, c}      {b, c}
  endif;
  a := c + 1              // {b, c}      {a, b, c}
done;
return c                  // {c}         empty
```

Réduction naïve

Définition

Soit $(x := a)^l$ un bloc d'affectation où $x \in \mathbb{V}$ et $l \in \mathbb{L}$. Si $x \notin LIVE_{out}[l]$, on dit que la variable x est morte.

Approche

Notons $\mathcal{A} = \mathcal{P}(\mathbb{V})^2$ et soit l'application

$$\Delta : \mathcal{A} \times Stm \longrightarrow Stm$$

On recalcule l'analyse à partir de $\perp$.

Exemple

```
a := 0;                  // empty      {a}
b := a;                  // {a}        {a, b}
while a < 100 do         // {a, b}     {a, b}
   if a = 2 then         // {a, b}     {a, b}
      c := a             // {a, b}     {b, c}
   else
      c := 2 * a;        // {a, b}     {b, c}
      d := b             // {b, c}     {b, c}
   endif;
   a := c + 1            // {b, c}     {a, b, c}
done;
return c                 // {c}        empty
```

Exemple

```
a := 0;                  // empty   {a}
b := a;                  // {a}     {a}
while a < 100 do         // {a}     {a}
  if a = 2 then          // {a}     {a}
    c := a               // {a}     {c}
  else
    c := 2 * a;          // {a}     {c}
    skip                 // {c}     {c}
  endif;
  a := c + 1             // {c}     {a, c}
done;
return c                 // {c}     empty
```

Exemple

```
a := 0;                 // empty   {a}
skip;                   // {a}     {a}
while a < 100 do        // {a}     {a}
  if a = 2 then         // {a}     {a}
    c := a              // {a}     {c}
  else
    c := 2 * a;         // {a}     {c}
    skip                // {c}     {c}
  endif;
  a := c + 1            // {c}     {a, c}
done;
return c                // {c}     empty
```

Incrémentalisation

- Mal optimisé : calcul d'un point fixe à partir de $\perp$ à chaque itération
- Est-il possible de réutiliser la précédente analyse ?
- La monotonie sera-t-elle conservée ?

Exemple

```
a := 0;              // empty      {a}
b := a + 1;          // {a}        {a, b}
c := 2 * b;          // {a, b}     {a}
return a             // {a}        empty
```

Exemple

```
a := 0;             // empty  {a}
b := a + 1;         // {a}    {a}
skip;               // {a}    {a}
return a            // {a}    empty
```

Remarque

Cette analyse de vivacité est plus petite que la précédente, on perd la monotonie !

Incrémentalisation

- Perte de la monotonie en prenant l'analyse précédente.
- Il faut alors pouvoir la réduire suffisament.
- Est-il possible d'y appliquer un filtre ?

Exemple

```
a := 0;                   // empty      {a}
b := a;                   // {a}        {a, b}
b := b + 3;               // {a, b}     {a, b}
while a < 100 do          // {a, b}     {a, b}
   if a = 2 then          // {a, b}     {a, b}
      c := a;             // {a, b}     {b, c}
      d := b              // {b, c}     {b, c}
   else
      c := 2 * a;         // {a, b}     {b, c}
      e := b              // {b, c}     {b, c}
   endif;
   a := c + 1             // {b, c}     {a, b, c}
done;
return c                  // {c}        empty
```

Exemple

```
a := 0;                 // empty  {a4, a5, a6, a8}
b := a;                 // {a4, a5, a6, a8}  {a4, a5, a6, a8, b3}
b := b + 3;             // {a4, a5, a6, a8, b3}  {a4, a5, a6, a8, b7, b9}
while a < 100 do        // {a4, a5, a6, a8, b7, b9}  {a5, a6, a8, b7, b9}
  if a = 2 then         // {a5, a6, a8, b7, b9}  {a6, a8, b7, b9}
    c := a;             // {a6, b7, b9}  {b7, b9, c10, c11}
    d := b              // {b7, b9, c10, c11}  {b7, b9, c10, c11}
  else
    c := 2 * a;         // {a8, b7, b9}  {b7, b9, c10, c11}
    e := b              // {b7, b9, c10, c11}  {b7, b9, c10, c11}
  endif;
  a := c + 1            // {b7, b9, c10, c11}  {a4, a5, a6, a8, b7, b9, c11}
done;
return c                // {c11}  empty
```

On travaille désormais sur le treillis $(\mathcal{P}(\mathbb{L} \times \mathbb{V}), \subseteq)$.

Notation

$$\mathcal{L} = \{l \in \mathbb{L} \mid (x := a)^l \text{ et } \{x\} \notin LIVE_{out}[l]\}$$

Si $s \in Stm$ une déclaration, on notera $s[\mathcal{L} \to skip]$ cette même déclaration, réduite aux blocs d'étiquette dans $\mathcal{L}$.

Incrémentalisation

On redéfinit l'analyse sur ce treillis.

$$\begin{aligned}
\mathit{gen} : \mathbb{L} \times \mathit{Block} &\longrightarrow \mathcal{P}(\mathbb{L} \times \mathbb{V}) \\
(l, x := a) &\longmapsto \mathit{vars}_a(l, a) \\
(l, \mathit{skip}) &\longmapsto \emptyset \\
(l, b) &\longmapsto \mathit{vars}_b(l, b) \\
(l, \text{return } a) &\longmapsto \mathit{vars}_a(l, a)
\end{aligned}$$

Incrémentalisation

$$
\begin{aligned}
\mathit{kill} : \mathit{Block} &\longrightarrow \mathcal{P}(\mathbb{L} \times \mathbb{V}) \\
x := a &\longmapsto \mathbb{L} \times \{x\} \\
\mathit{skip} &\longmapsto \emptyset \\
b &\longmapsto \emptyset \\
\text{return } a &\longmapsto \emptyset
\end{aligned}
$$

Théorème

Soient $s \in Stm$, et $s' = s[\mathcal{L} \rightarrow skip]$ une réduction sur l'ensemble d'étiquettes $\mathcal{L}$. Soient μ_s et $\mu_{s'}$ les point fixes minimaux respectifs des deux déclarations. Alors il existe $I_{\mathcal{L}}$ tel que $\forall l \in \mathbb{L}$,

$$\mu_s[l] - I_{\mathcal{L}} \subseteq \mu_{s'}[l]$$

est un pré-point fixe de $\mu_{s'}[l]$.

Remarque

On prendra $I_{\mathcal{L}} = \mathcal{L} \times \mathbb{V}$

Exemple

Reprenons le premier exemple :

```
a := 0;              // empty          {a2, a4}
b := a + 1;          // {a2, a4}       {a4, b3}
c := 2 * b;          // {a4, b3}       {a4}
return a             // {a4}           empty
```

Remarque

$\forall l \in \mathbb{L}$ on a $(l, c) \notin \{(4, a)\}$ donc c est morte.

Exemple

`a := 0;`	`// empty`	$\{a_2, a_4\}$
`b := a + 1;`	`//` $\{a_2, a_4\}$	$\{a_4\}$
`skip;`	`//` $\{a_4\}$	$\{a_4\}$
`return a`	`//` $\{a_4\}$	`empty`

Exemple

`a := 0;`	`// empty`	$\{a_4\}$
`skip;`	`//` $\{a_4\}$	$\{a_4\}$
`skip;`	`//` $\{a_4\}$	$\{a_4\}$
`return a`	`//` $\{a_4\}$	`empty`

Généralisation

Avec un prédicat sur la vivacité des variables.

```
a := 0;
#if {b} ∈ LIVE_in[3]
  b := a + 1;
#if {c} ∈ LIVE_in[4]
  c := 2 * b;
return a
```

Généralisation

Considérons le programme

```c
int div(int x)
{
  return x/32;
}
```

Son code assembleur, produit par GCC

```asm
div:
  mov     eax, DWORD PTR [rbp-4]
  lea     edx, [rax+31]
  test    eax, eax
  cmovs   eax, edx
  sar     eax, 5
  ret
```

Généralisation

En supposant l'entier toujours positif

```
div:
  mov     eax, DWORD PTR [rbp-4]
  shr     eax, 5
  ret
```

Posons alors le prédicat

$$P : v \longmapsto [\![v > 0 \vee v = 0]\!]^{B}(\sigma)$$