frama C

Software Analyzers

Alias: pointeurs espionnés en série

un analyseur de pointeurs « léger » pour C

Tristan Le Gall[2], Jan Rochel[2], Florian Faissole[1],
Julien Signoles[2], Denis Cousineau[1]

30 janvier 2024 @ JFLA 2024

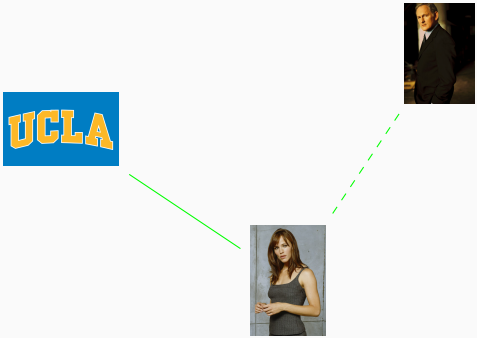
1 〉 Mitsubishi Electric R&D Centre Europe, Rennes, France (MERCE)
2 〉 Université Paris-Saclay, CEA, List, Palaiseau, France

cea list

université PARIS-SACLAY

MITSUBISHI ELECTRIC
Changes for the Better

ALIAS




CENTRAL INTELLIGENCE AGENCY
UNITED STATES OF AMERICA


ALIAS

SD6


UCLA

Alliance des 12

SD6

UCLA

Alliance des 12

SD6

UCLA

CENTRAL INTELLIGENCE AGENCY
UNITED STATES OF AMERICA

ALIAS

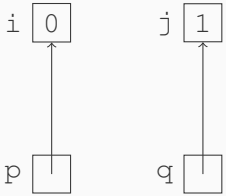
Alliance des 12

SD6

EXEMPLE : ALIAS, MAY-ALIAS

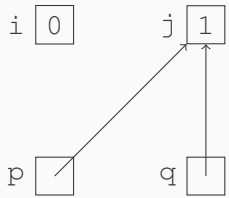

```
int i = 0, j = 1;
```

EXEMPLE : ALIAS, MAY-ALIAS




```
int i = 0, j = 1;

int *p = &i, *q = &j;
```

EXEMPLE : ALIAS, MAY-ALIAS

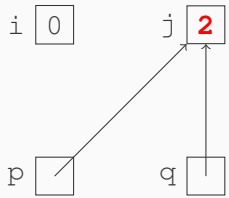



```
int i = 0, j = 1;

int *p = &i, *q = &j;

p = q;
```

Si deux pointeurs `p` et `q` pointent vers la même case de mémoire on dit :

- `p` est un alias de `q`.
- `p` et `q` sont aliasés.
- `p` et `q` sont des alias.
- `p` et `q` sont en alias.

Exemple : alias, may-alias

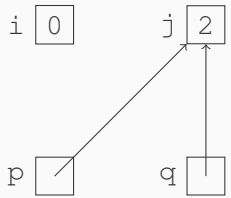



```
int i = 0, j = 1;

int *p = &i, *q = &j;

p = q;

*q = 2;    // *p == 2
```

Si deux pointeurs `p` et `q` pointent vers la même case de mémoire on dit :

- `p` est un alias de `q`.
- `p` et `q` sont aliasés.
- `p` et `q` sont des alias.
- `p` et `q` sont en alias.

EXEMPLE : ALIAS, MAY-ALIAS

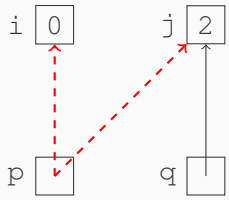



```
int i = 0, j = 1;

int *p = &i, *q = &j;

p = q;

*q = 2;
```

Pertinence de l'analyse d'alias / de pointeurs :

- la correction des compilateurs
- l'optimisation de code
- l'analyse de programmes concurrents

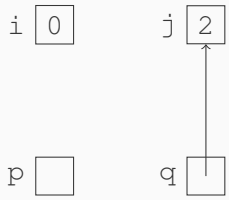
EXEMPLE : ALIAS, MAY-ALIAS




```
int i = 0, j = 1;

int *p = &i, *q = &j;

if random_cond() p = q;

*q = 2;
```

Indécidable : est-ce que `p` et `q` sont aliasés ?

EXEMPLE : ALIAS, MAY-ALIAS

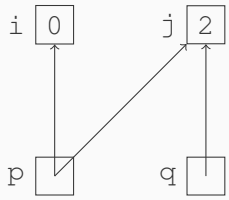



```
int i = 0, j = 1;

int *p = &i, *q = &j;

if random_cond() p = q;

*q = 2;
```

Indécidable : est-ce que `p` et `q` sont aliasés ?

- Sous-approximation : graphe de *must-pointeurs*
  $\Rightarrow$ Résultat : *must-alias* $\{\{\}\}$

EXEMPLE : ALIAS, MAY-ALIAS




```
int i = 0, j = 1;

int *p = &i, *q = &j;

if random_cond() p = q;

*q = 2;
```

Indécidable : est-ce que `p` et `q` sont aliasés ?

- Sous-approximation : graphe de *must-pointeurs*
  ⇒ Résultat : *must-alias* $\{\{\}\}$
- Sur-approximation : graphe de *may-pointeurs*
  ⇒ Résultat : *may-alias* $\{\{\mathtt{p}, \mathtt{q}\}\}$

FRAMA-C : plateforme pour analyser et transformer du code C

- un noyau : parser, AST, typage, analyse dataflow, etc.
- des greffons pour des analyses et transformations variées

FRAMA-C : plateforme pour analyser et transformer du code C

- un noyau : parser, AST, typage, analyse dataflow, etc.
- des greffons pour des analyses et transformations variées

greffon EVA : analyse statique basé sur l'interprétation abstraite

- peut effectuer une analyse d'alias mais c'est une analyse « lourde »
- nécessite de calculer des invariants de boucle, des calculs numériques
- requière une certaine expertise

FRAMA-C : plateforme pour analyser et transformer du code C

- un noyau : parser, AST, typage, analyse dataflow, etc.
- des greffons pour des analyses et transformations variées

greffon EVA : analyse statique basé sur l'interprétation abstraite

- peut effectuer une analyse d'alias mais c'est une analyse « lourde »
- nécessite de calculer des invariants de boucle, des calculs numériques
- requière une certaine expertise

greffon ALIAS : on veut une analyse « légère »

- « pousse-bouton » : aucune expertise nécessaire
- rapide, pas de calculs numériques, avec une assez bonne précision
- API à grain fin : analyse partielle possible

Le greffon Alias

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (***context-sensitive***)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (***context-sensitive***)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

context-sensitive :

{{x,a}}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (***context-sensitive***)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

context-sensitive :

$\{\{x,a\},\{y,b\}\}$

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (***context-sensitive***)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

context-sensitive :

{{x,a},{y,b}}

context-*in*sensitive :

{{x,a,y,b}}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (***flow-sensitive***)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

flow-sensitive :

{}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (***flow-sensitive***)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

flow-sensitive :

{} {}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (***flow-sensitive***)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

flow-sensitive :

{{x,a}} {} {}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (***flow-sensitive***)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

flow-sensitive :

{{x,a},{y,b}} {{x,a}} {} {}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (***flow-sensitive***)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int *a, *b;
  int *x, *y;
  alias (x, a);
  alias (y, b);
}
```

flow-sensitive :

{{x,a},{y,b}} {{x,a}} {} {}

flow-*in*sensitive : {{x,a},{y,b}}

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (***field-sensitive***)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  struct {int *a, *b;} x, y;

  alias (x.a, y.a);
  alias (x.b, y.b);
}
```

Le greffon Alias

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (***field-sensitive***)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  struct {int *a, *b;} x, y;

  alias (x.a, y.a);
  alias (x.b, y.b);
}
```

field-sensitive :

$\{\{\texttt{x.a}, \texttt{y.a}\}, \{\texttt{x.b}, \texttt{y.b}\}\}$

field-*in*sensitive :

$\{\{\texttt{x.a}, \texttt{x.b}, \texttt{y.a}, \texttt{y.b}\}\}$

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (***array-insensitive***)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int[2] *x;
  int[2] *y;
  alias (x[0], y[0]);
  alias (x[1], y[1]);
}
```

LE GREFFON ALIAS

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (***array-insensitive***)
  - pas d'arithmétique de pointeurs

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int[2] *x;
  int[2] *y;
  alias (x[0], y[0]);
  alias (x[1], y[1]);
}
```

array-sensitive :

$\{\{x[0], y[0]\}, \{x[1], y[1]\}\}$

array-*in*sensitive :

$\{\{x[0], x[1], y[0], y[1]\}\}$

Le greffon Alias

- basé sur l'algorithme de Steensgaard
- analyse *dataflow*
- sensibilités :
  - appels de fonction (*context-sensitive*)
  - position (*flow-sensitive*)
  - structures (*field-sensitive*)
- insensibilités :
  - tableaux (*array-insensitive*)
  - pas d'arithmétique de pointeurs
- traite un sous-ensemble du langage C (adapté au cas d'usage de MERCE)

```
void alias(int *p, int *q) {
  p = q;
}

int main () {
  int[2] *x;
  int[2] *y;
  alias (x[0], y[0]);
  alias (x[1], y[1]);
}
```

Pourquoi Steensgaard ?

- plus rapide (mais moins précis) que l'algorithme d'Andersen
- profite des fonctionalités offertes par le noyau de Frama-C
- permet d'utiliser des structures efficaces (OCamlgraph, Union-Find)

Exemple

```
int* f(int *fst, int *snd, int **i1, int **i2, int b) {
  fst = *i1;
  if (b) {*snd = *fst; return *i2;}
  else   {fst = *i2; return *i1;}
}

void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Exemple

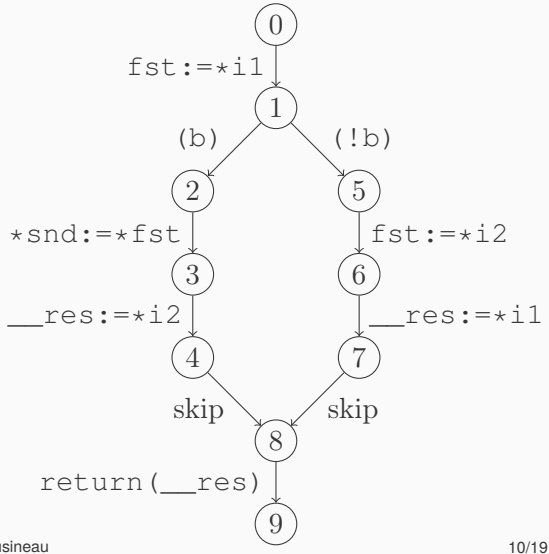
```
int* f(int *fst, int *snd, int **i1, int **i2, int b) {
  fst = *i1;
  if (b) {*snd = *fst; return *i2;}
  else   {fst = *i2; return *i1;}
}
```

Normalisation par Frama-C

```
int* f(int *fst, int *snd, int **i1, int **i2, int b) {
  int* __res;
  fst = *i1;
  if (b) {*snd = *fst; __res = *i2;}
  else   {fst = *i2; __res = *i1;}
  return __res;
}
```

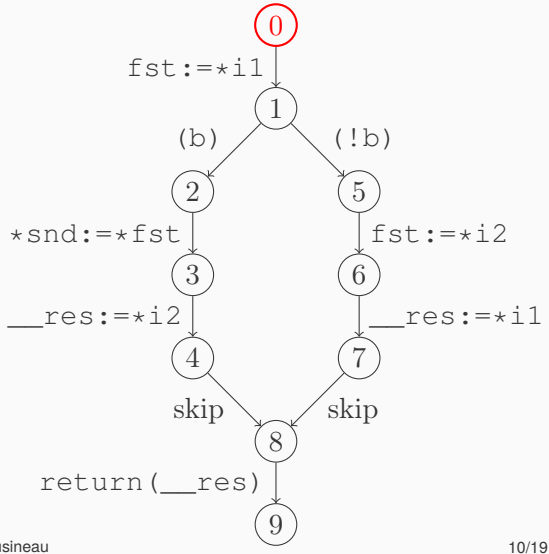
Graphe de flot de contrôle (CFG)

```
int* f(int *fst, int *snd, ...) {
  int* __res;
  fst = *i1;
  if (b) *snd = *fst; __res = *i2;
  else   fst = *i2; __res = *i1;
  return __res;
}
```



Analyse du flux des données : graphes d'alias

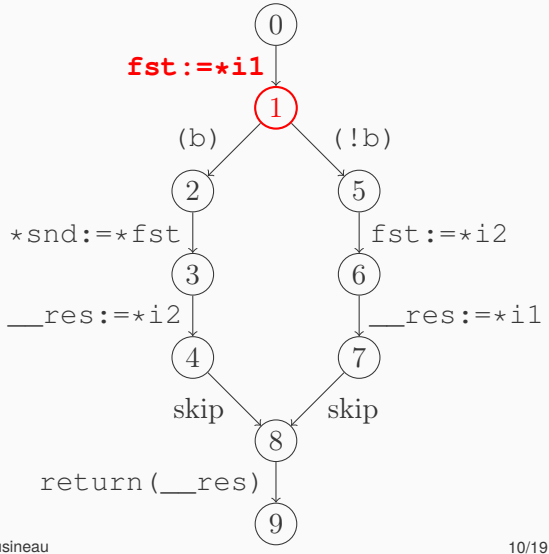
état initial : graphe vide



Analyse du flux des données : graphes d'alias

function transfert(lv, e, G)
  if ispointer(lv) **then**
    let $n_1, G := $ **find_or_create**(lv, G) **in**
    let $n_2, G := $ **find_or_create**(e, G) **in**
    let $n, G := $ **fusion_rec**(n_1, n_2, G) **in**
    return G
  else return G

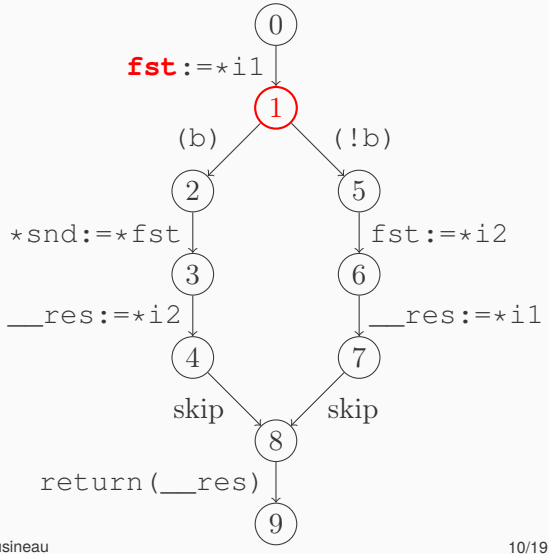
pour une affectation $lv := e$



Analyse du flux des données : graphes d'alias

function transfert(lv, e, G)
  if ispointer(lv) **then**
    let $n_1, G := $ **find_or_create**(lv, G) **in**
    let $n_2, G := $ **find_or_create**(e, G) **in**
    let $n, G := $ **fusion_rec**(n_1, n_2, G) **in**
    return G
  else return G

{fst}

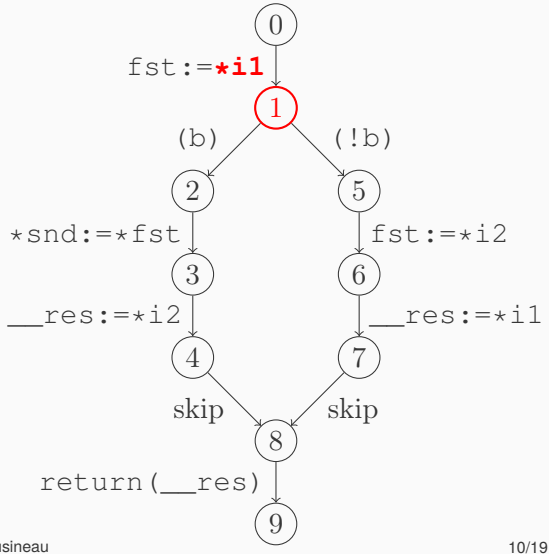


Analyse du flux des données : graphes d'alias

```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

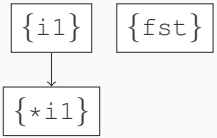
{fst}

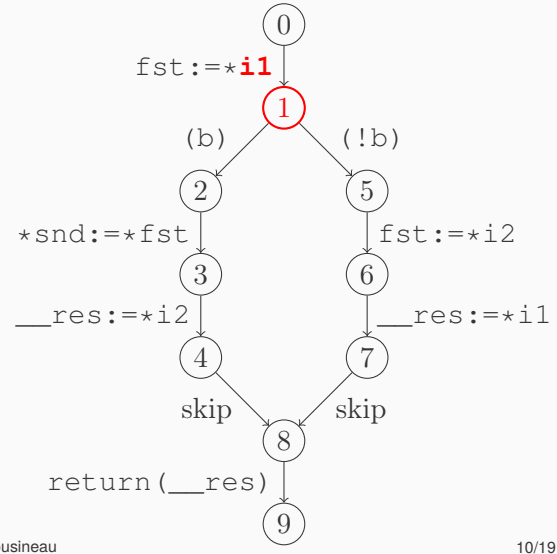
{*i1}



Analyse du flux des données : graphes d'alias

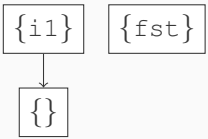
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

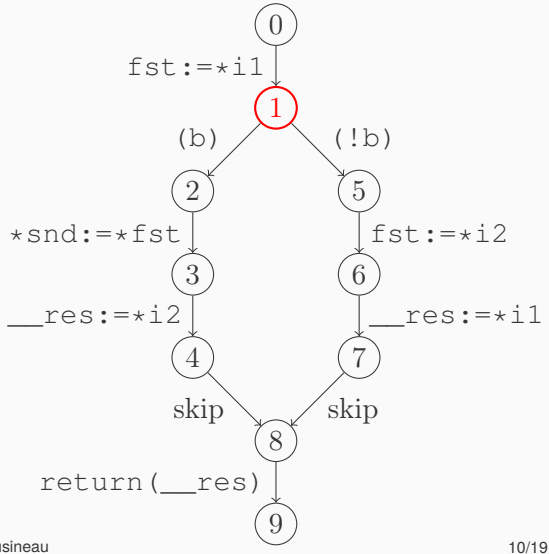




Analyse du flux des données : graphes d'alias

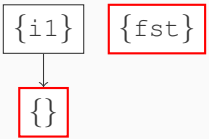
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

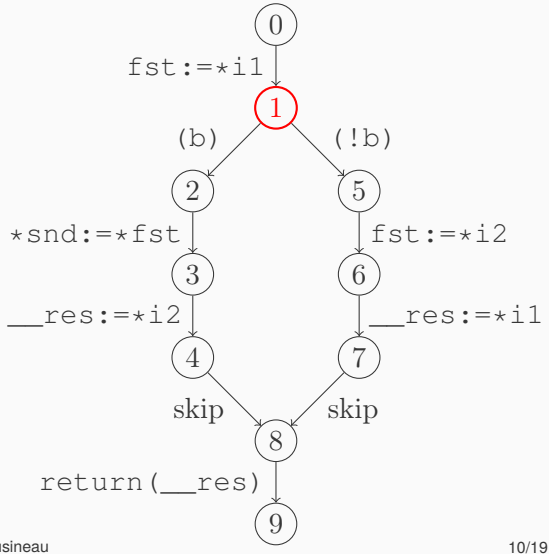




Analyse du flux des données : graphes d'alias

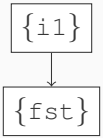
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

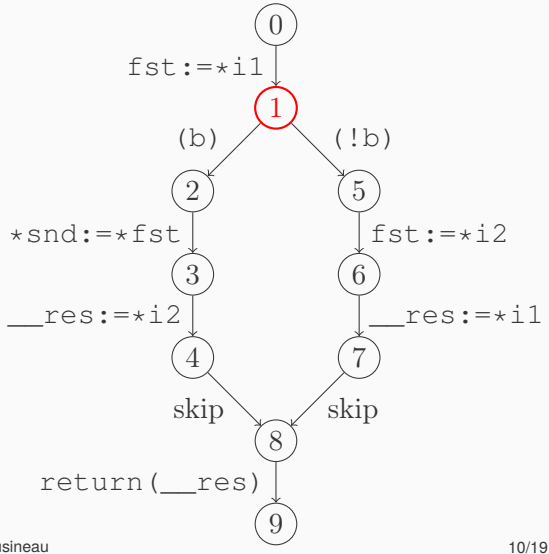




Analyse du flux des données : graphes d'alias

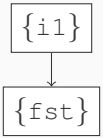
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

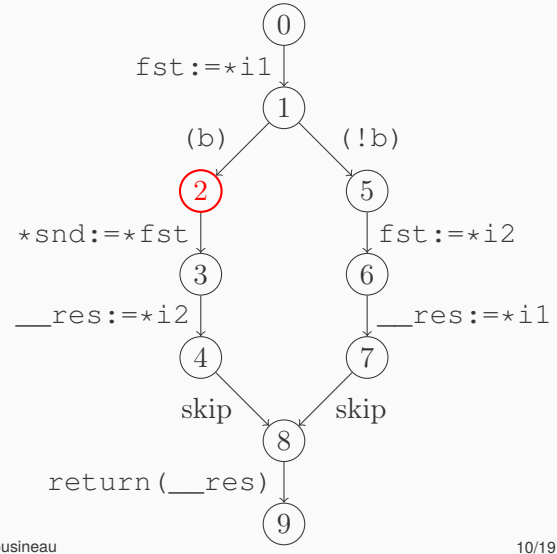




Analyse du flux des données : graphes d'alias

```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```



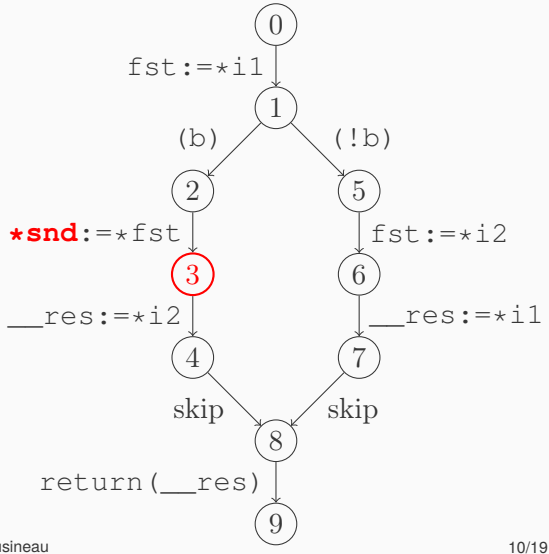


Analyse du flux des données : graphes d'alias

```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

```
int* f(int *fst, int *snd,
       int **i1, int **i2, int b) {
```

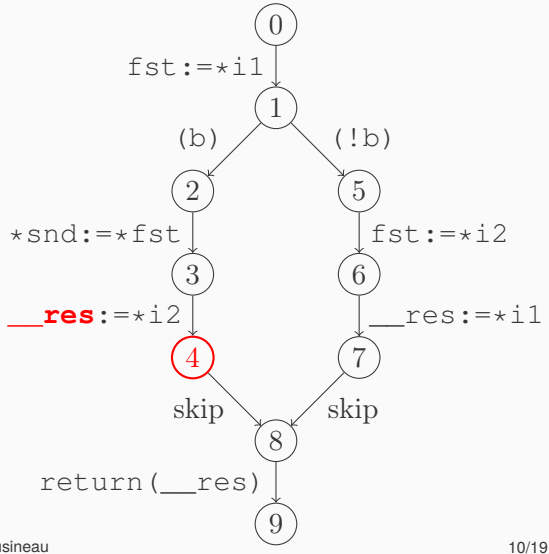
affectation scalaire
$\Rightarrow$ aucun changement au graphe



Analyse du flux des données : graphes d'alias

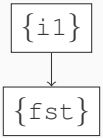
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

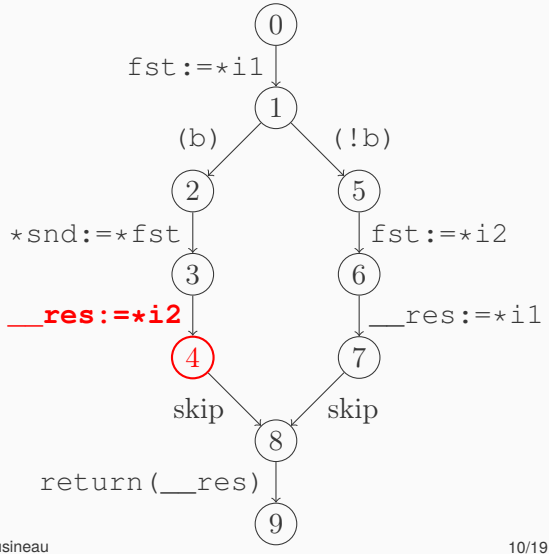
```
int* f(int *fst, int *snd,
       int **i1, int **i2, int b) {
int* __res;
```



Analyse du flux des données : graphes d'alias

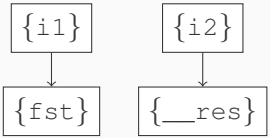
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

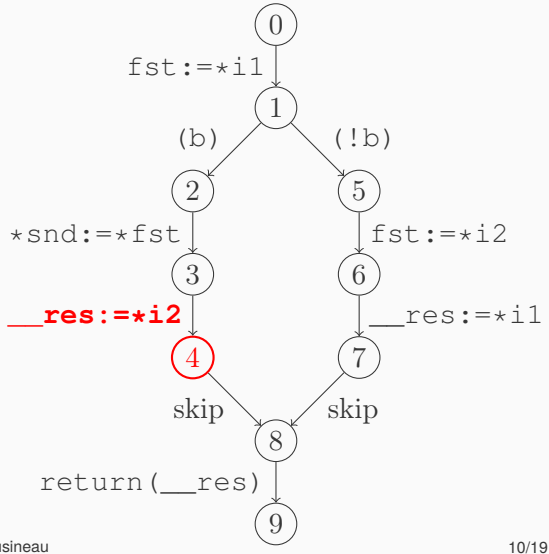




Analyse du flux des données : graphes d'alias

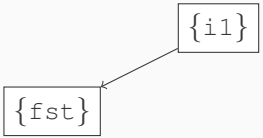
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

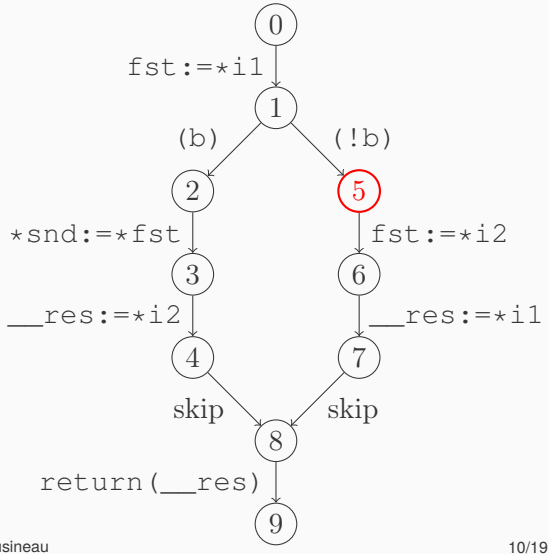




Analyse du flux des données : graphes d'alias

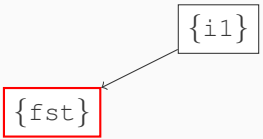
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

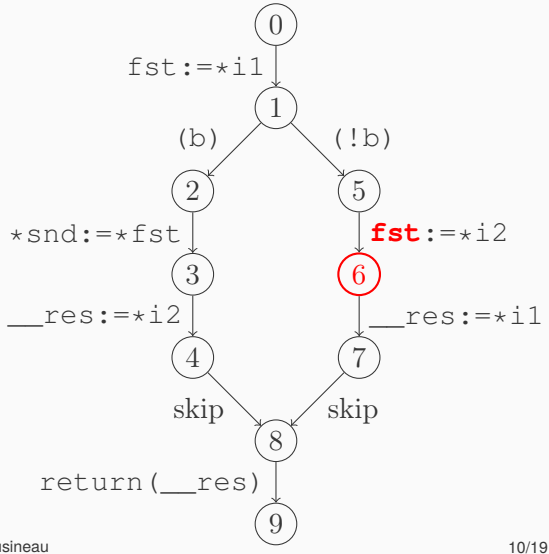




Analyse du flux des données : graphes d'alias

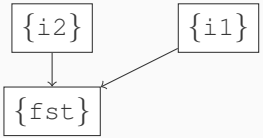
```
function transfert(lv, e, G)
  if ispointer(lv) then
    let n1, G := find_or_create(lv, G) in
    let n2, G := find_or_create(e, G) in
    let n, G := fusion_rec(n1, n2, G) in
    return G
  else return G
```

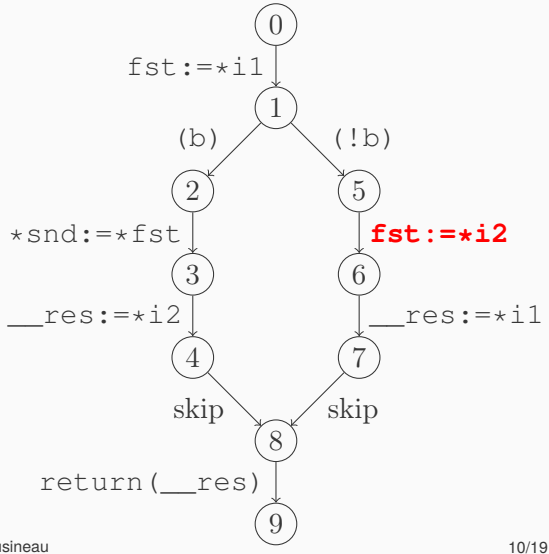




Analyse du flux des données : graphes d'alias

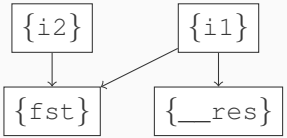
function transfert(lv, e, G)
  if ispointer(lv) **then**
    let $n_1, G := $ **find_or_create**(lv, G) **in**
    let $n_2, G := $ **find_or_create**(e, G) **in**
    let $n, G := $ **fusion_rec**(n_1, n_2, G) **in**
    return G
  else return G

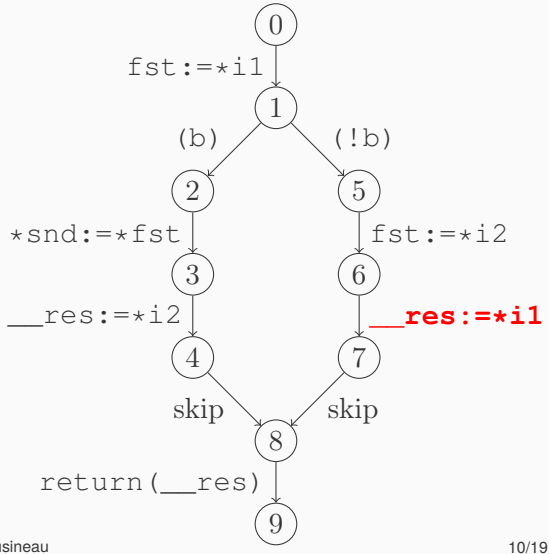




Analyse du flux des données : graphes d'alias

function transfert(lv, e, G)
  if ispointer(lv) **then**
    let $n_1, G := $ **find_or_create**(lv, G) **in**
    let $n_2, G := $ **find_or_create**(e, G) **in**
    let $n, G := $ **fusion_rec**(n_1, n_2, G) **in**
    return G
  else return G

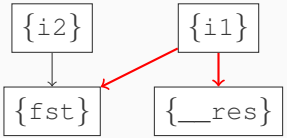


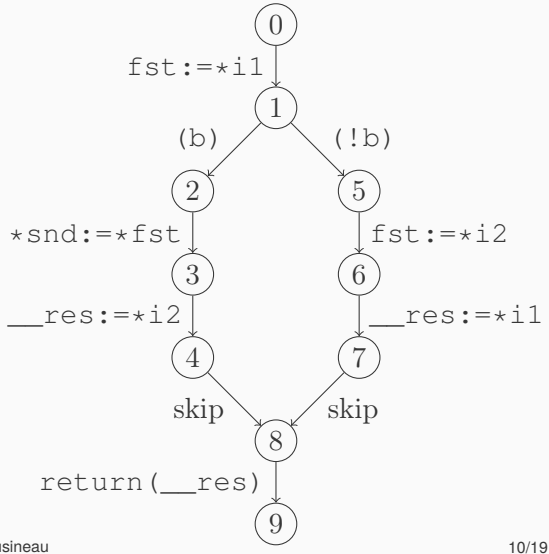


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud

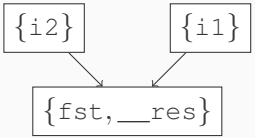


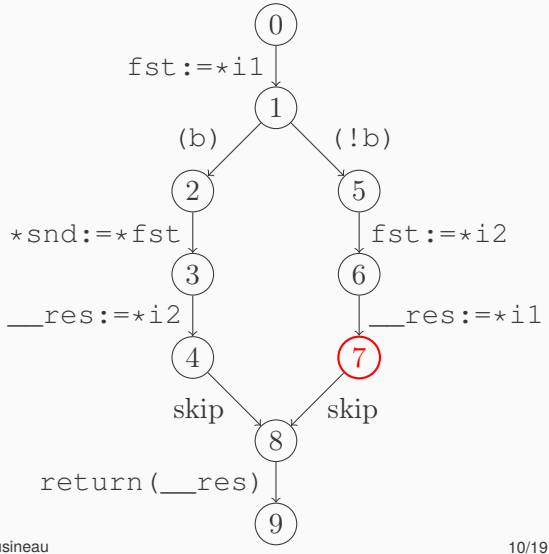


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud

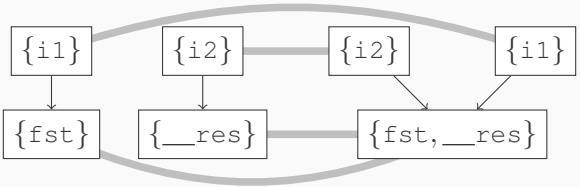


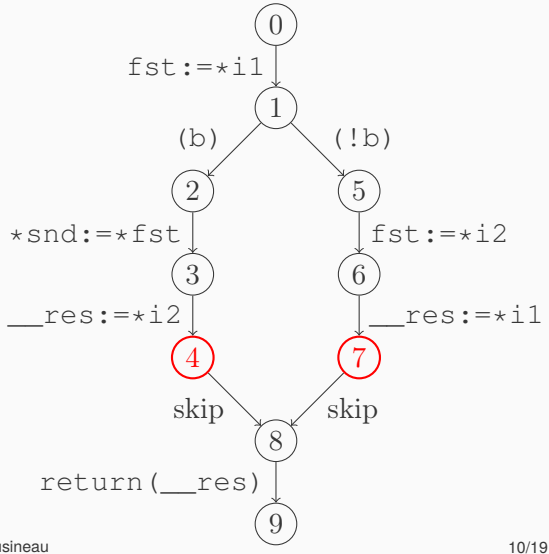


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud

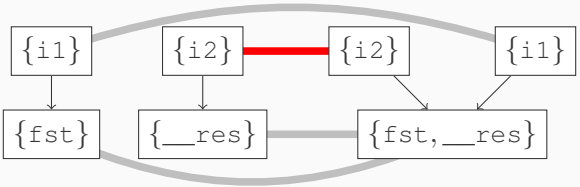


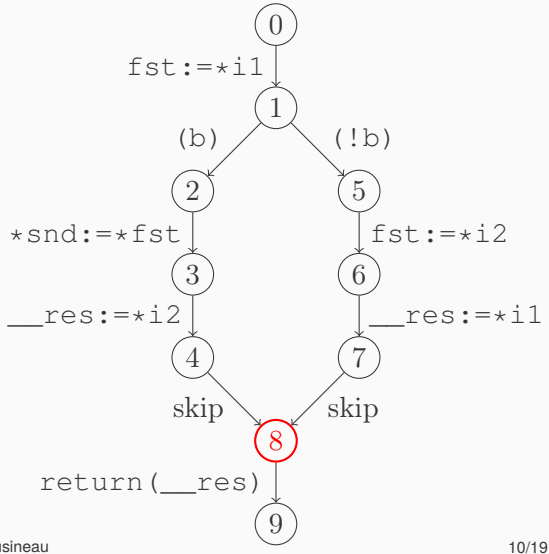


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud
- invariant 2 : une seule occurrence par variable

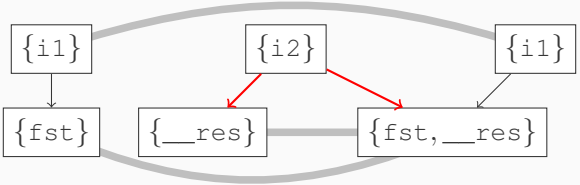


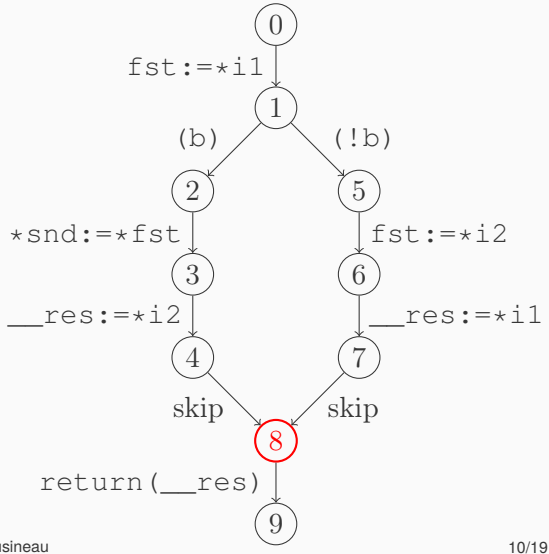


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud
- invariant 2 : une seule occurrence par variable

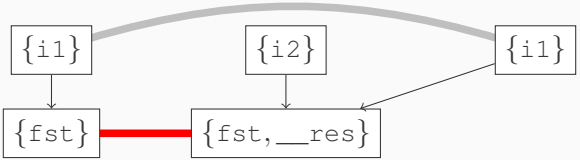


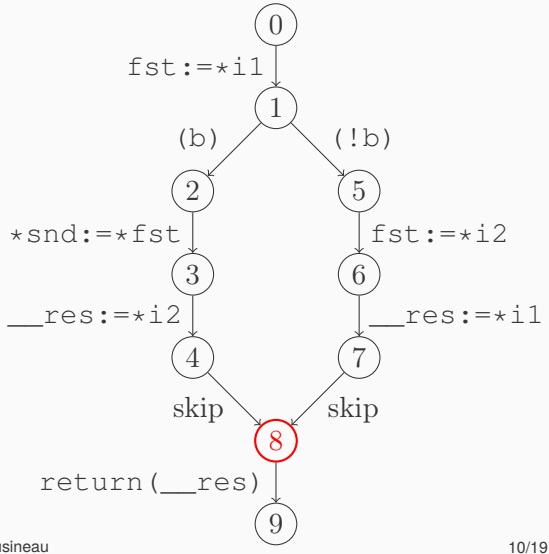


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud
- invariant 2 : une seule occurrence par variable

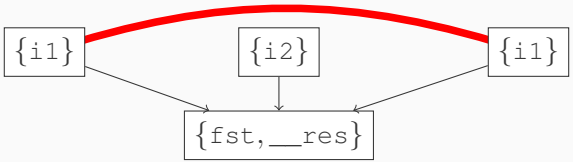


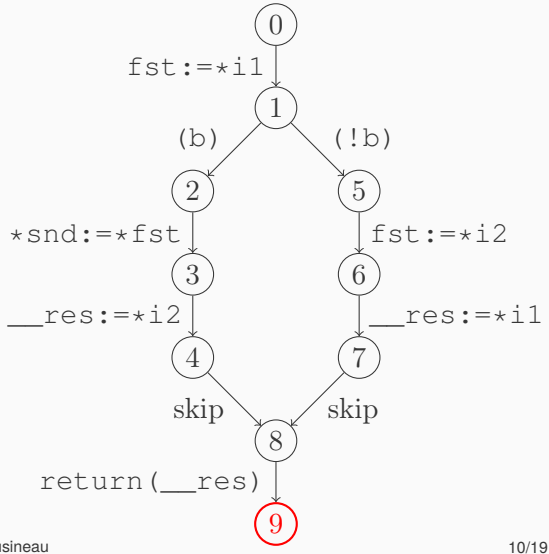


Analyse du flux des données : graphes d'alias

Invariants du graphe de pointeurs :

- invariant 1 : un seul successeur par nœud
- invariant 2 : une seule occurrence par variable

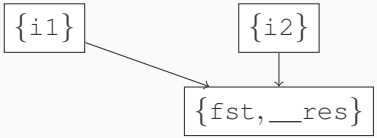


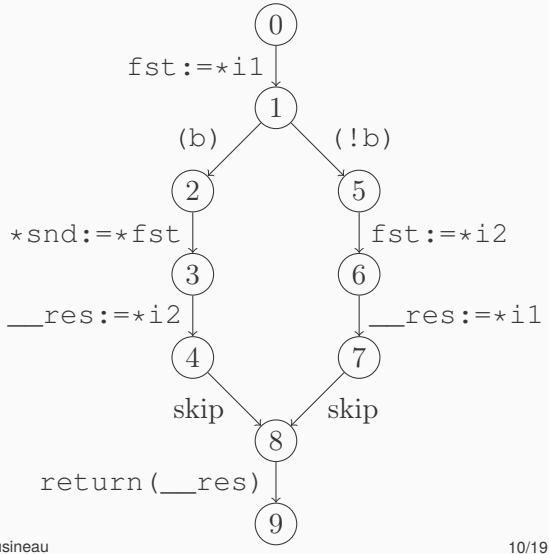


Analyse du flux des données : graphes d'alias

Résumé pour la fonction `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`
- avec le graphe :

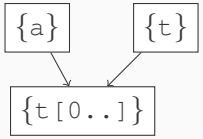




```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

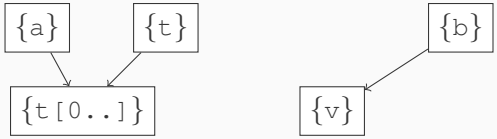
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



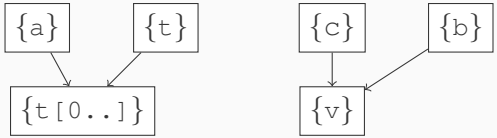
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



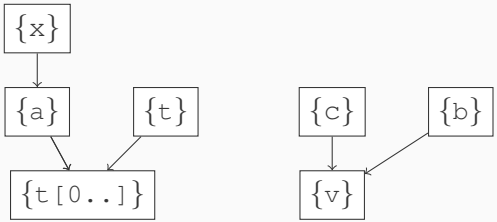
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



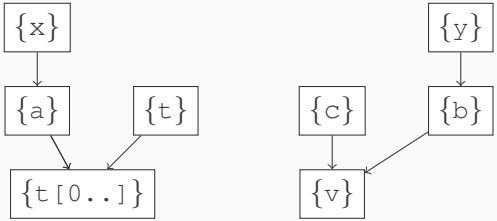
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int ***x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



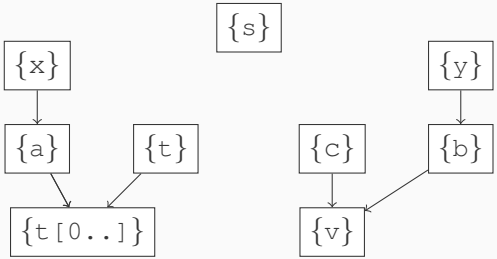
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



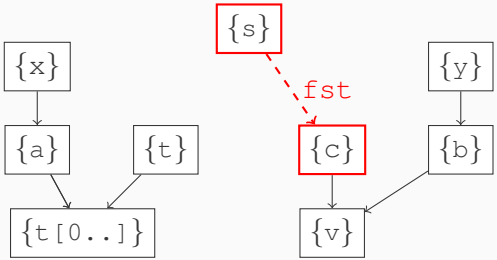
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

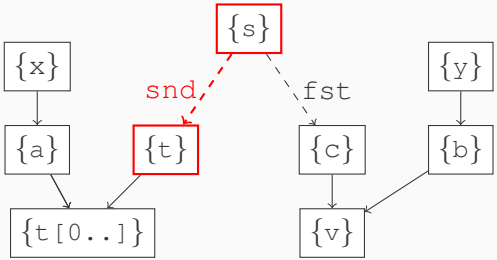


Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

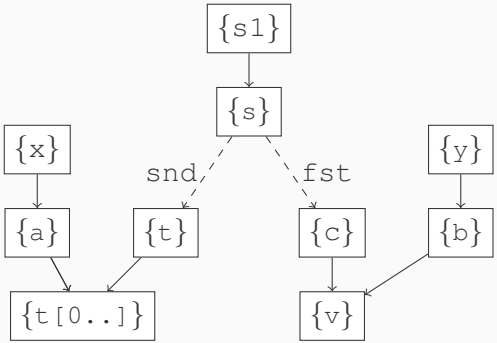


```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



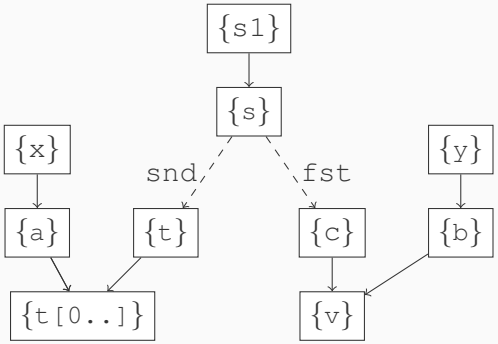
Interprocédurabilité

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

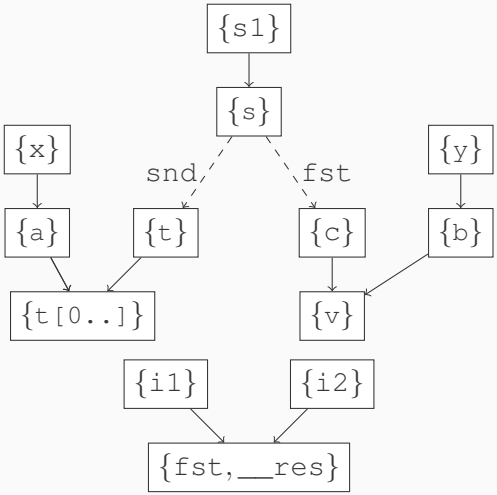


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

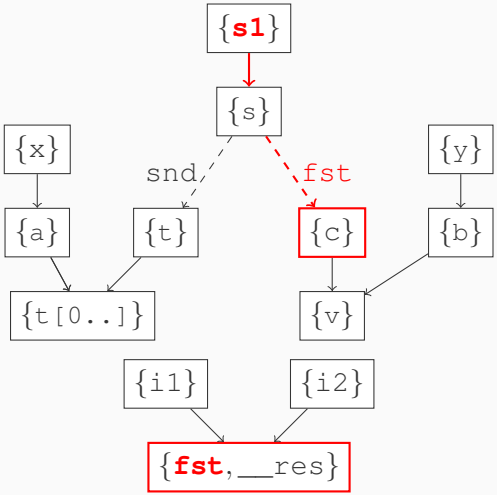


Interprocéduralité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : **`fst`**, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

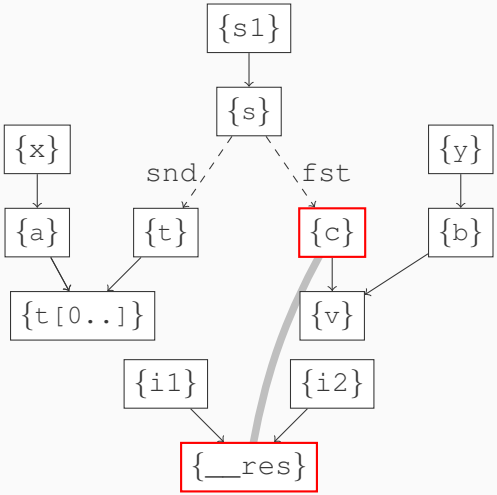


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : **`fst`**, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

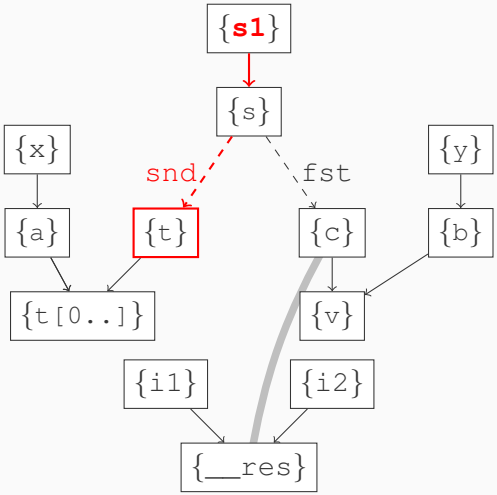


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, **`snd`**, `i1`, `i2`, `b`
- variable de retour : `__res`

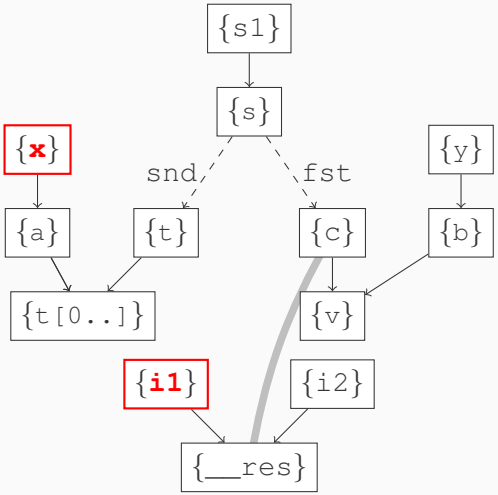


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, **`i1`**, `i2`, `b`
- variable de retour : `__res`

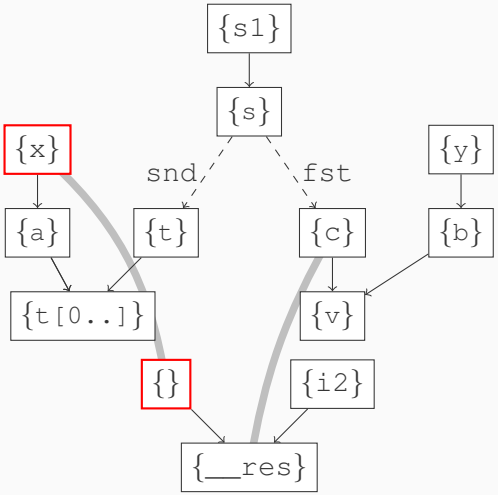


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, **`i1`**, `i2`, `b`
- variable de retour : `__res`

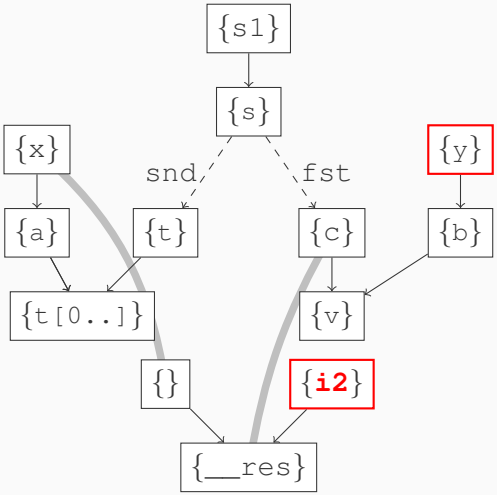


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, **`i2`**, `b`
- variable de retour : `__res`

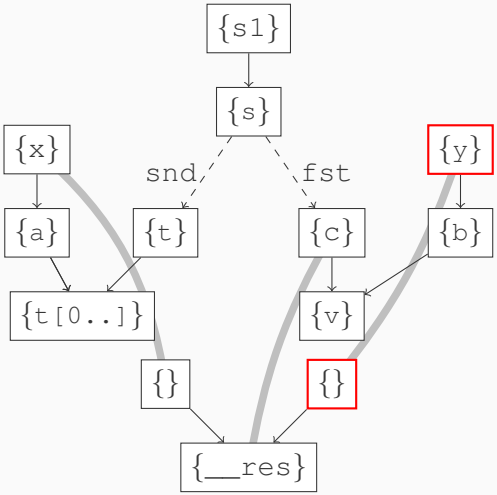


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, **`i2`**, `b`
- variable de retour : `__res`

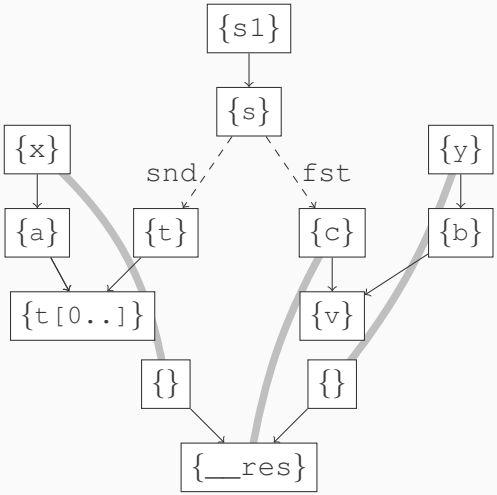


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, **`b`**
- variable de retour : `__res`

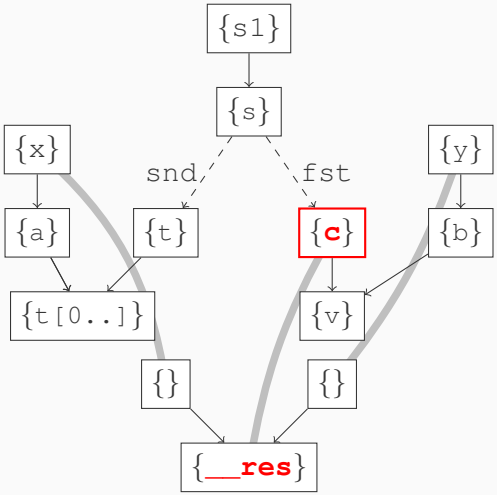


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : **`__res`**

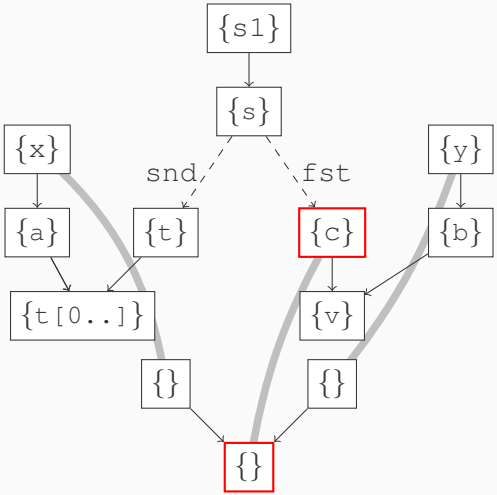


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

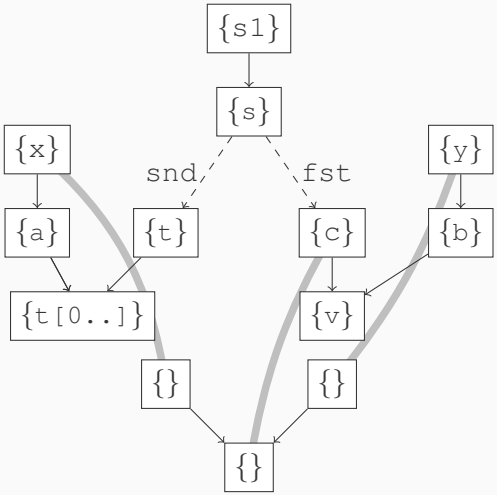


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

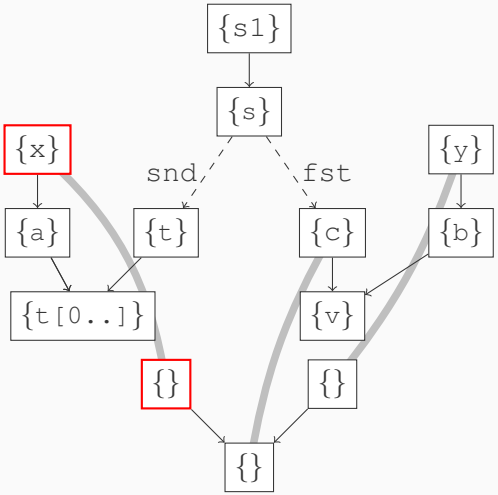


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

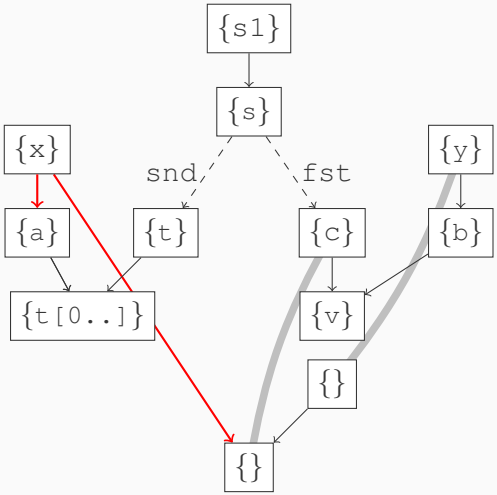


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

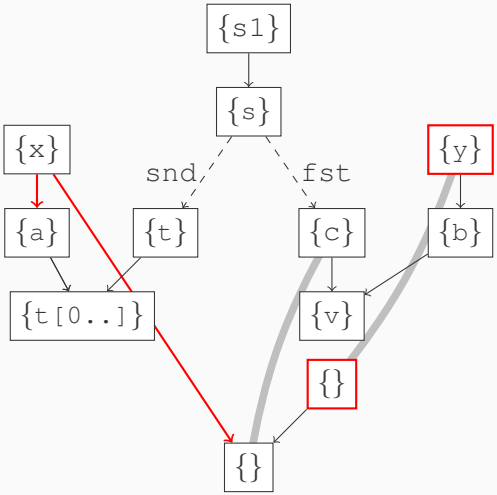


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

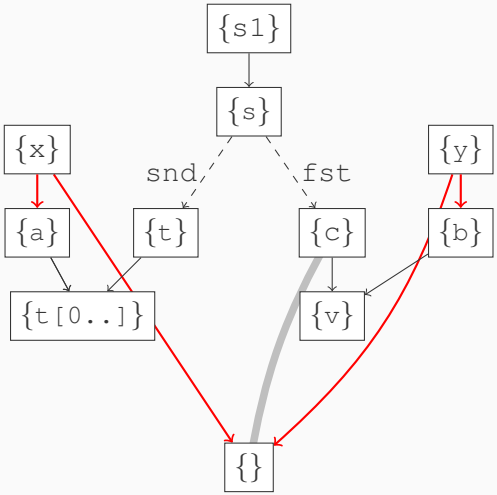


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

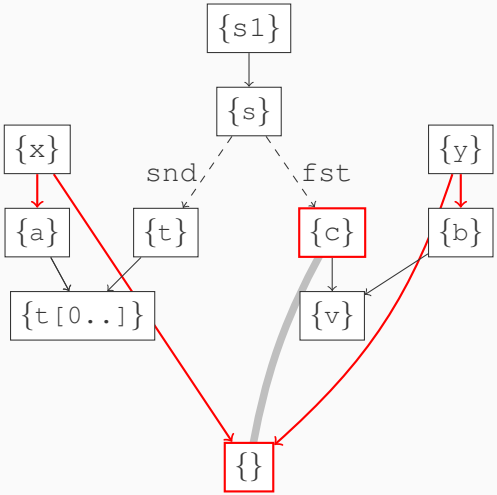


Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

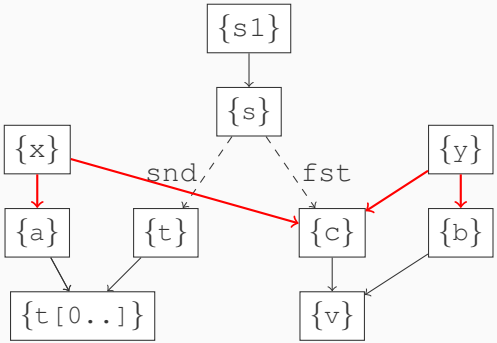
Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`



Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

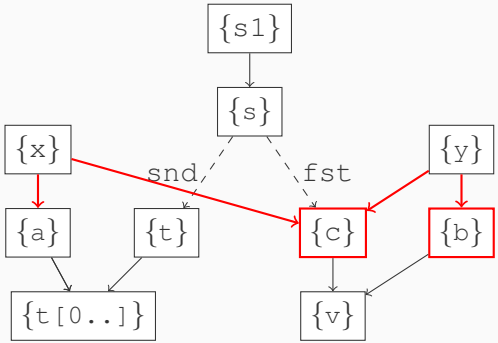


Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

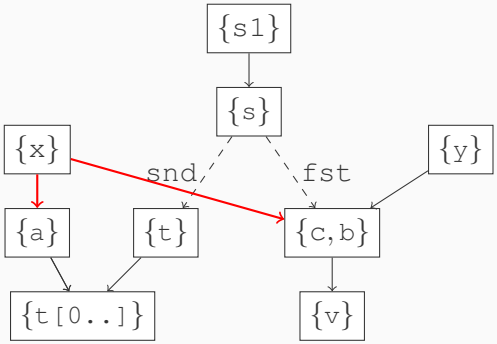


Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

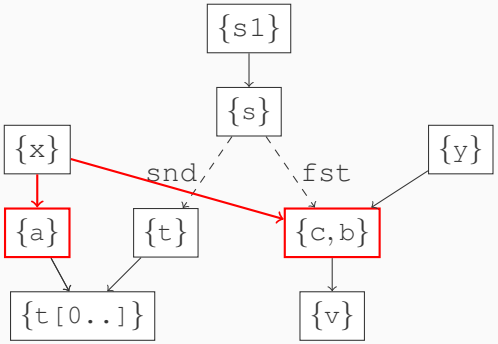


Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

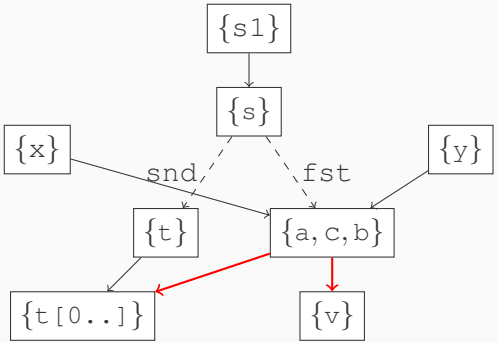


Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

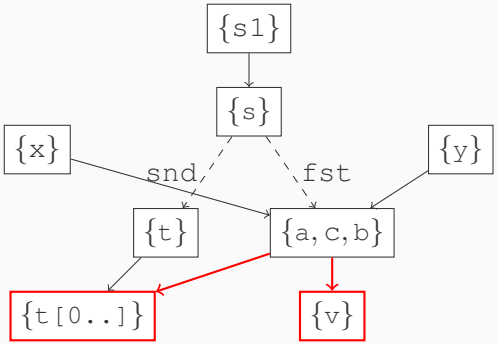


Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```



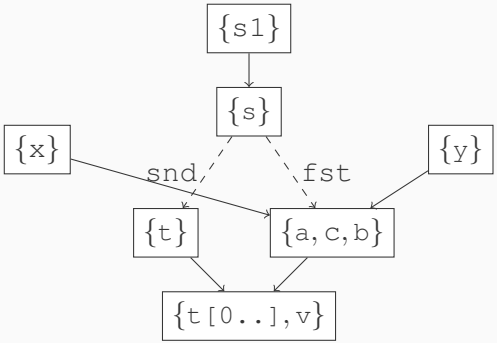
Résumé de `f` :

- paramètres formels : `fst`, `snd`, `i1`, `i2`, `b`
- variable de retour : `__res`

Interprocédurabilité : instantiation du résumé

```
void main(void) {
  int v = 9, t[3] = {0,1};
  int *a = &t[1], *b = &v, *c = &v;
  int **x = &a, **y = &b;
  struct {int *fst; int *snd;}
    s = {c, t}, *s1 = &s;
  c = f(s1->fst, s1->snd, x, y, 0);
}
```

Ensemble des may-alias : $\{\{x, y\}, \{t, a, c, b\}\}$



- Steensgaard comme analyse dataflow
- associe aux nœuds du CFG des graphes de pointeurs
- calcul des graphes de pointeurs :
  - les affectations génèrent des nœuds et des arêtes
  - les affectations scalaires sont sans effet
  - des arêtes spéciales pour les champs de structures
  - deux invariants qui assure une bonne performance
  - procède par fusion de classes d'équivalence

Open source case studies [1]

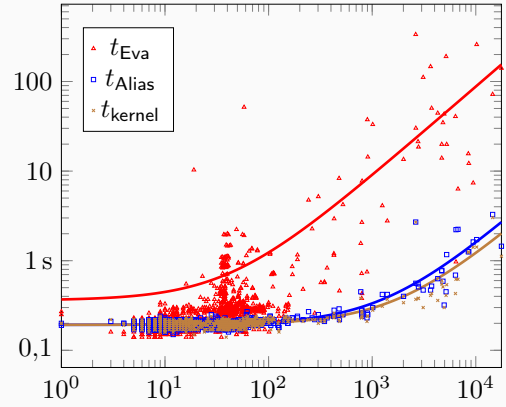
- collection de 36 projets open-source
- 270000 lignes de code C

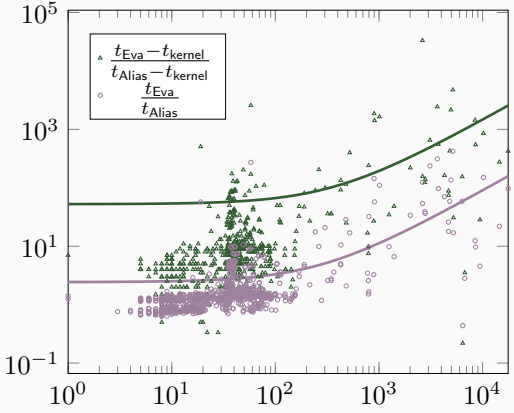
Comparaison avec Eva :

- temps d'exécution
- consommation de mémoire
- exactitude : est-ce que Alias trouve trop d'alias ?
- complétude : est-ce qu'il y en a que Alias ne voit pas ?

1. https://git.frama-c.com/pub/open-source-case-studies

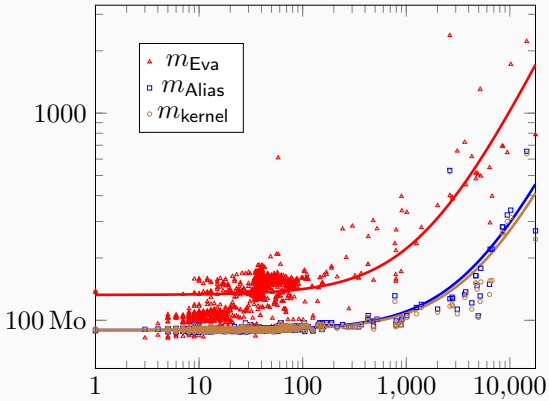
Expérimentation : temps d'exécution

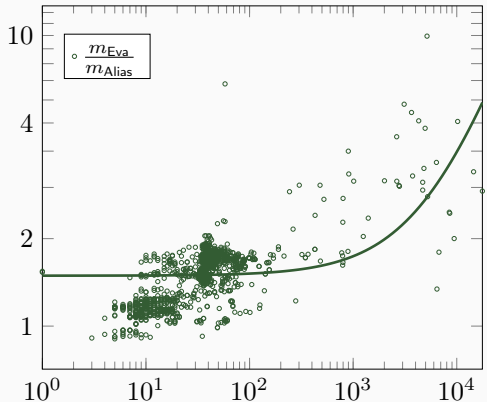




accélération ≈ 256

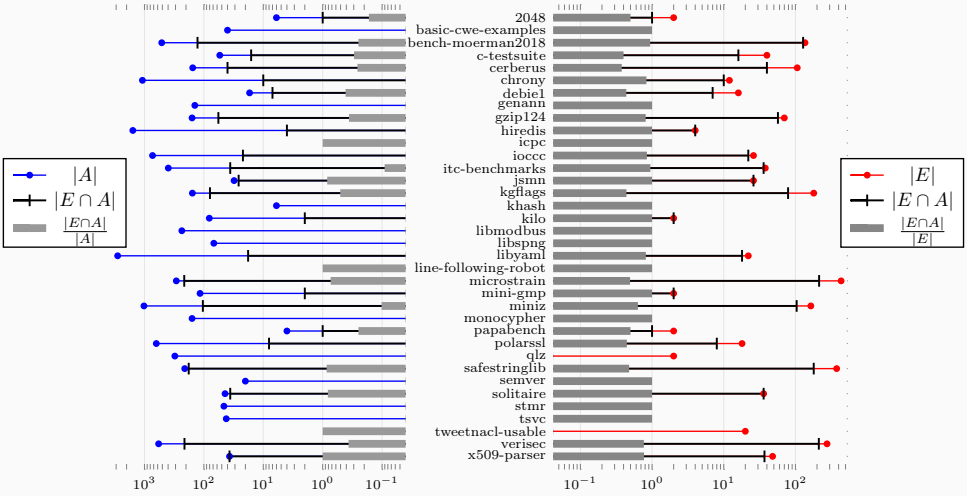
Expérimentation : consommation de mémoire





réduction de l'empreinte de mémoire $\approx 3{,}6$

Expérimentation : exactitude / complétude



exactitude $\approx 0{,}29$ complétude $\approx 0{,}77$

Applications en vue

- Génération automatique de programmes concurrents (MERCE)
  - enjeux : protégér par des verrous des variables accédées par des différentes tâches
  - Alias : identification des variables accédées par une tâche
- rCFI : Intégrité du flot de contrôle avec attestation à distance
  - enjeux : vérifier en temps d'exécution qu'un programme correspond au CFG
  - Alias : prendre en compte les pointeurs de fonctions dans le CFG
- Optimisation pour la vérification à l'exécution avec E-ACSL [2]
  - utiliser Alias pour une analyse de flot de donnees permettant de décider quelles zones mémoire E-ACSL doit monitorer
- Optimisation pour la vérification déductive avec WP
  - recherche en cours pour définir de nouveaux modèles mémoires permettant la vérification automatique en présence d'alias

2. C. Sánchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstić, J. M. Lourenço, D. Nickovic, G. J. Pace, J. Rufino, J. Signoles, D. Traytel, and A. Weiss. *A Survey of Challenges for Runtime Verification from Advanced Application Domains (Beyond Software)*. Formal Methods in System Design, August 2019.

- nouvelle analyse d'alias pour C basé sur Steensgaard
- greffon pour Frama-C
- rapide et facile à utiliser
- supporte une partie de C
- implémentation incomplète, à maturer
- des applications en vue

Annexe : constructions C non-supportées

- fonctions (mutuellement) récursives
- fonctions variadique définies par l'utilisateur
- fonctions sans implémentation
- appels indirects (via des pointeurs de fonction)
- code assembleur
- instructions `longjmp` and `setjmp`
- casts hétérogènes
- types union
- allocations de mémoire dynamiques (sauf au début du programme)
- instructions `goto` complexes