DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Graphe de dépendances

Petit Poucet style

M. Blanc, F. Desclaux, C. Leman, C. Mougey, A. Vincent

Direction des applications militaires

3 juin 2016

Problématique

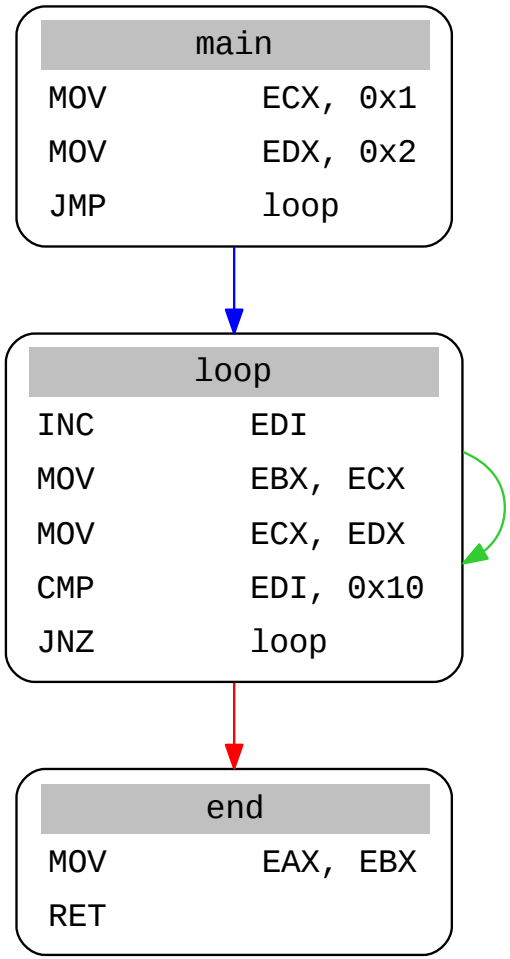
Analyse de flot de données : *data slicing*

Problème initial : quelles sont les opérations du programme qui participent à la valeur d'une variable donnée ?

Utilisation

- Analyse de malware
- Recherche de vulnérabilités
- Analyse de code

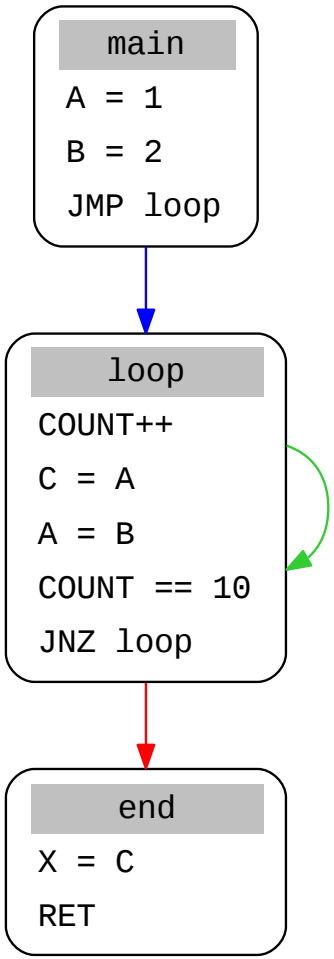
Exemple simplifié



Qui participe à la valeur finale d'`EAX` ?

- Ça s'appelle le *slicing* / *backward data tainting* / *whatever-ing*…
- …et ça commence à avoir 20 ans [Weiser, père et fils]

Exemple simplifié



Qui participe à la valeur finale de X ?

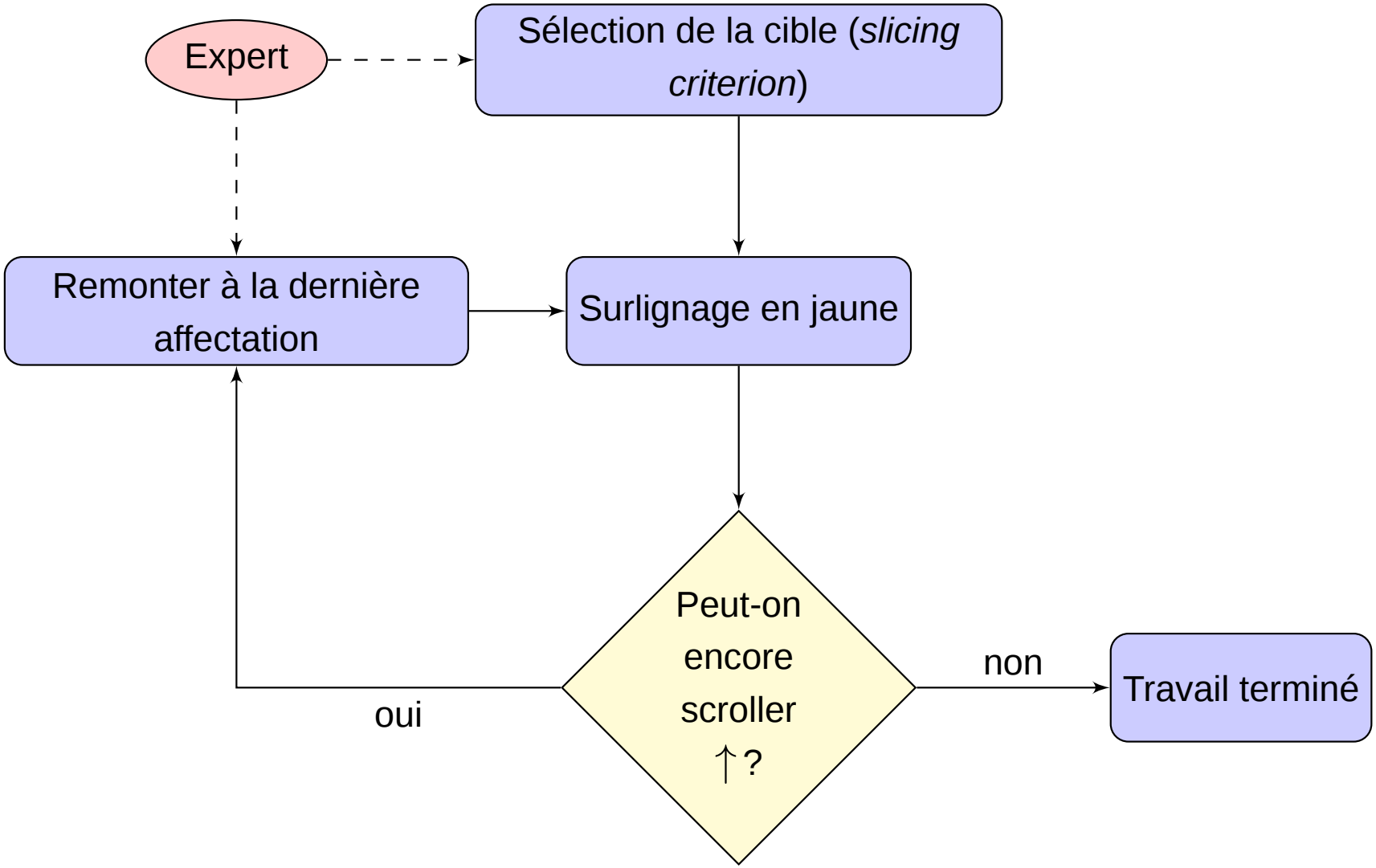
- Ça s'appelle le *slicing* / *backward data tainting* / *whatever-ing*…
- …et ça commence à avoir 20 ans [Weiser, père et fils]

Slicing

Tel le petit poucet, nous allons devoir retrouver notre chemin en déposant et en retrouvant nos cailloux

Analyse manuelle sous IDA

Méthode du baroudeur de binaire



Exemple : analyse de code

Retrouver la source de la taille d'un *memcpy*

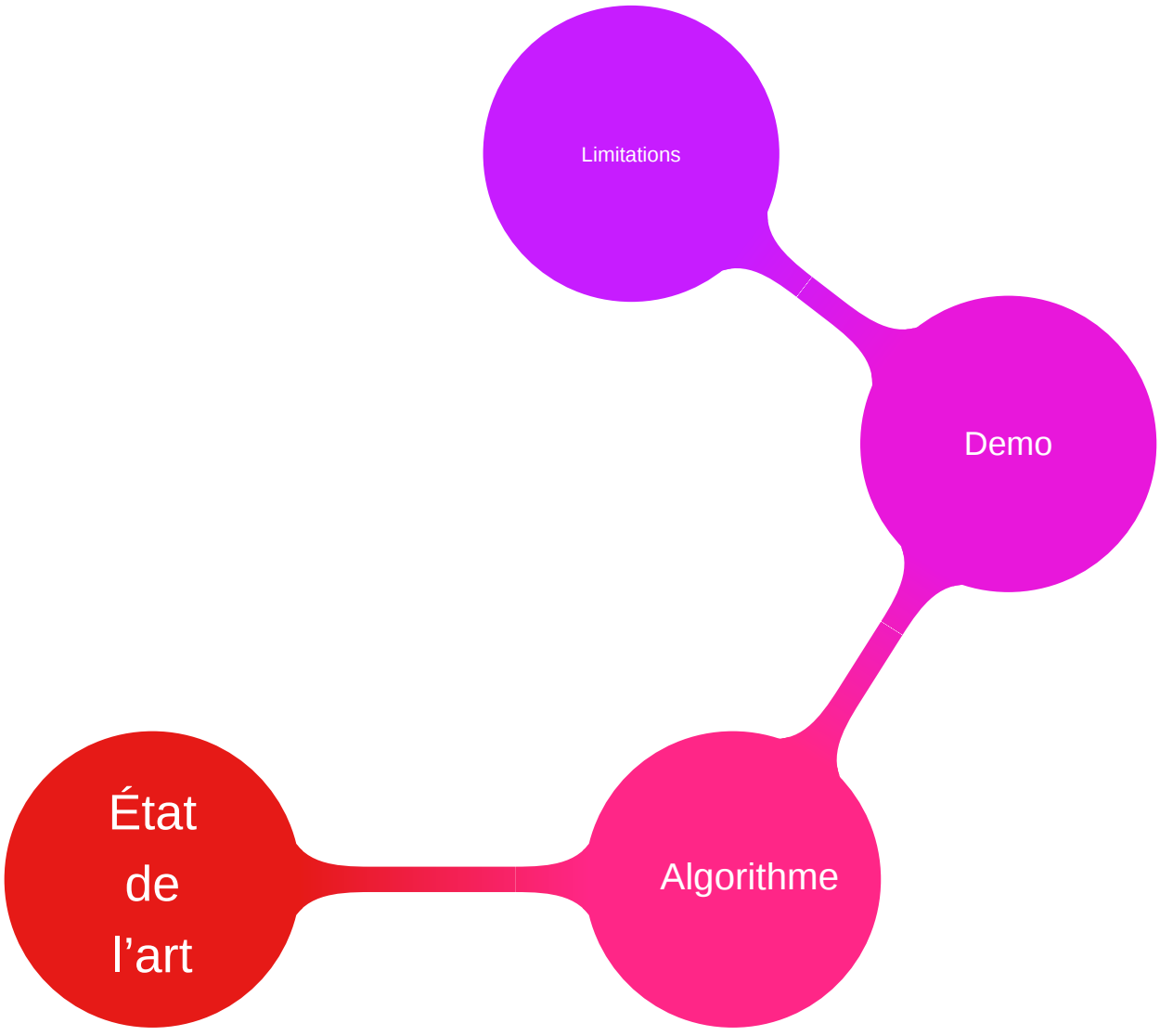
Fonction 0x401691, recherche des sources de EDI *depuis 0x4016F6*

Analyse manuelle

Inconvénients

- Passage à l'échelle difficile
- Sujet à l'erreur
- Chronophage

Plan



Angr

Angr : `angr.io`

- “Framework for analyzing binaries”
- Loader de binaire (ELF, PE)
- Multi architecture (ARM / AArch64, x86 32/64, Mips 32/64, PPC 32/64)
- Moteur d’exécution symbolique
- Méthodes d’analyse statique

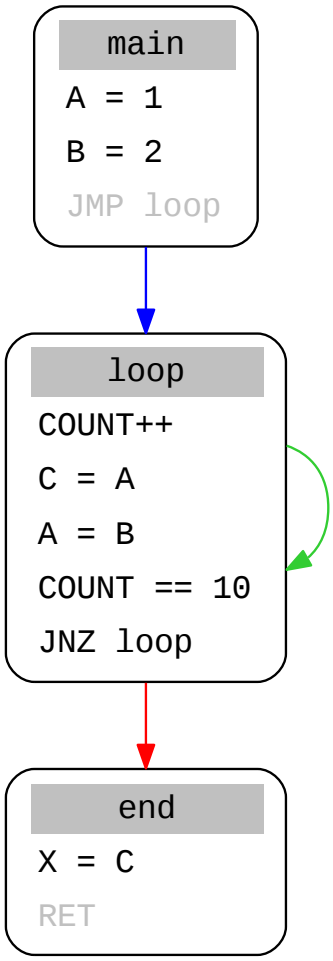
Use-define chains

Use-define chains

- “BackwardSlice”
- Statique, sans distinction de chemin d’exécution
- Efficace (de l’ordre de $\mathcal{O}(n + e)$)

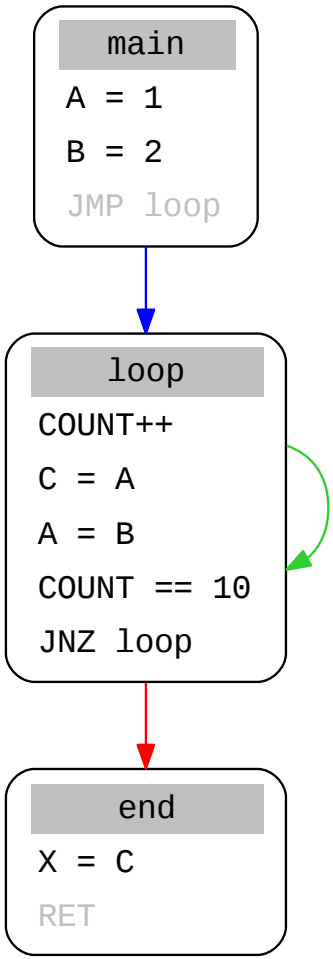
```
project = angr.Project(filename,
                       load_options={'auto_load_libs':False})
cfg = project.analyses.CFG(context_sensitivity_level=2, keep_state=True)
cdg = project.analyses.CDG(cfg)
ddg = project.analyses.DDG(cfg)

target_node = cfg.get_any_node(addr)
bs = project.analyses.BackwardSlice(cfg, cdg=cdg, ddg=ddg,
                                    targets=[(target_node, statement)])
```

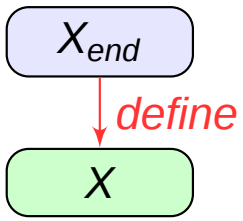
Use-define chains : Graphe de dépendances



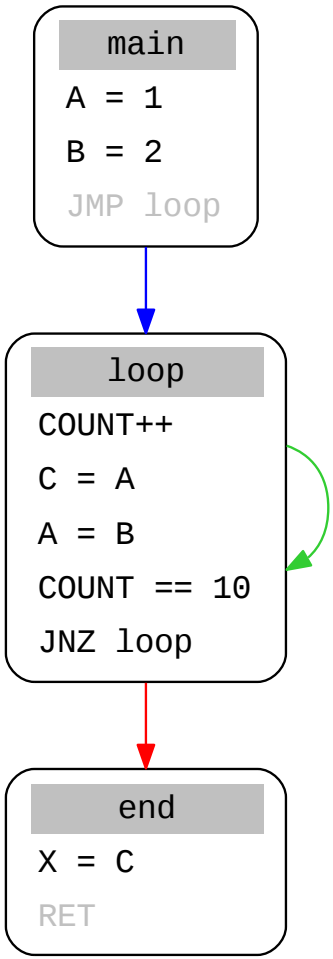
X

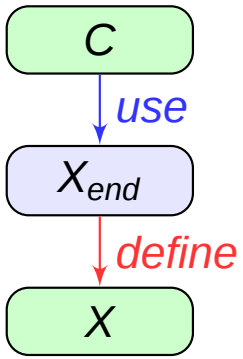
Use-define chains : Graphe de dépendances



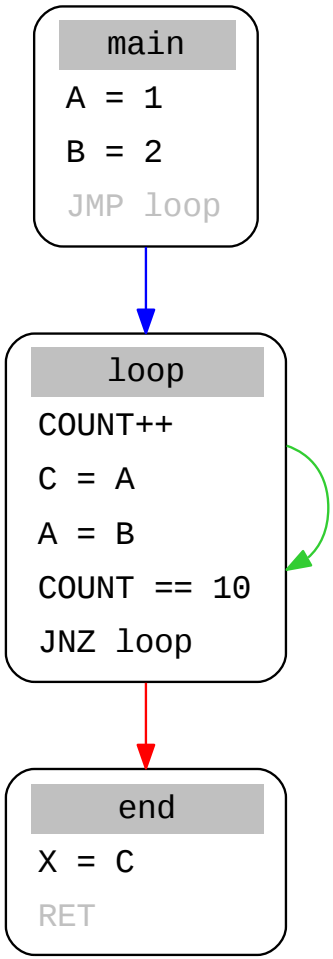


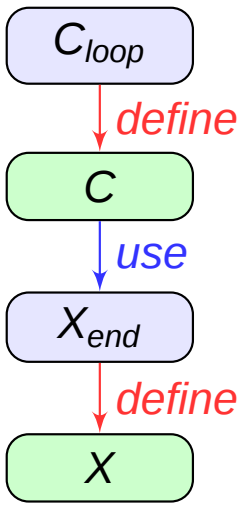
Use-define chains : Graphe de dépendances



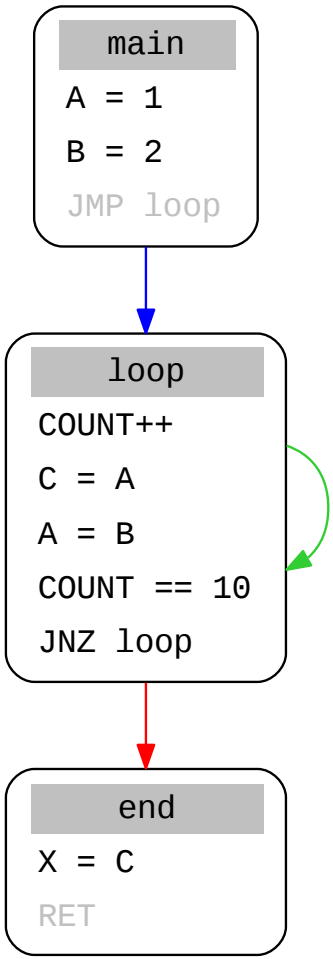


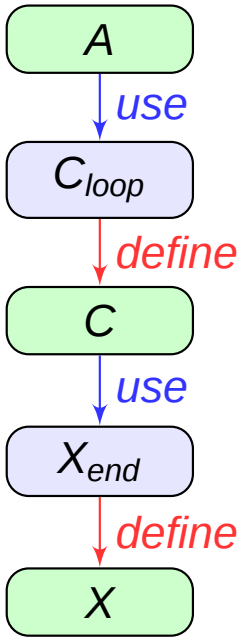
Use-define chains : Graphe de dépendances



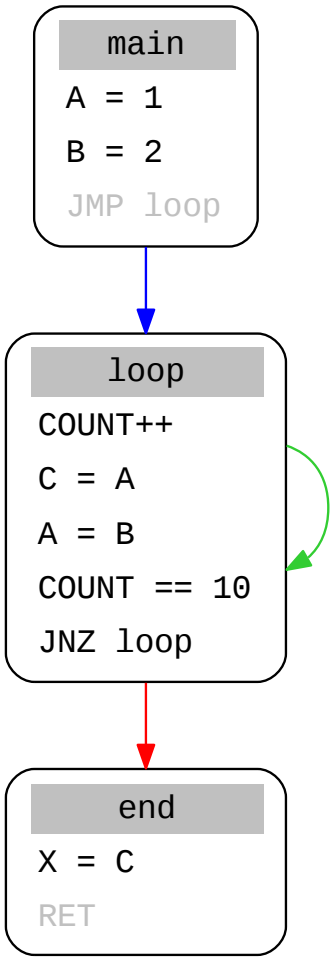


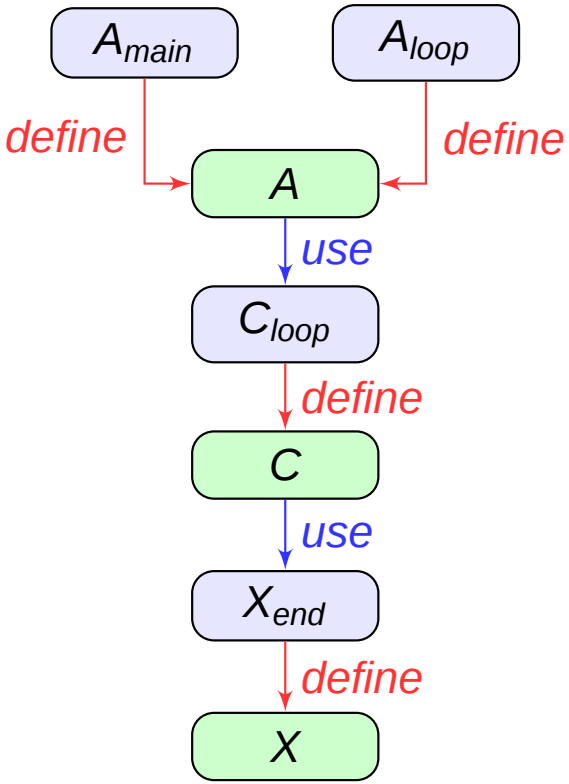
Use-define chains : Graphe de dépendances



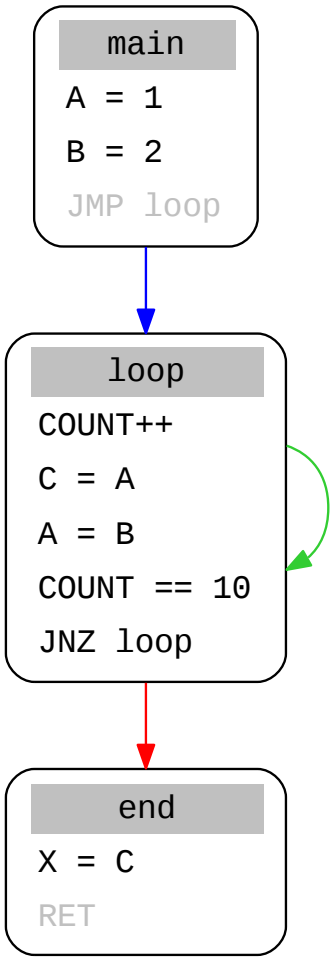


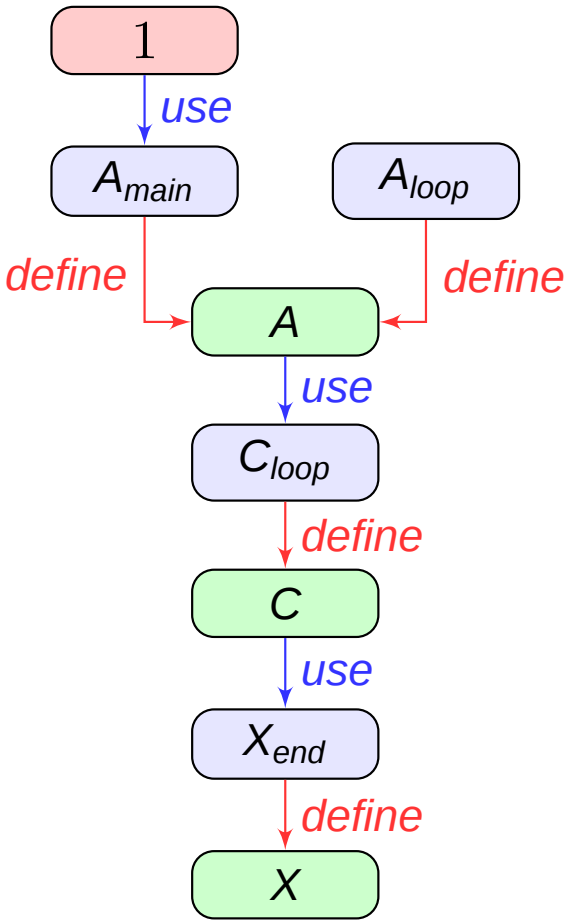
Use-define chains : Graphe de dépendances



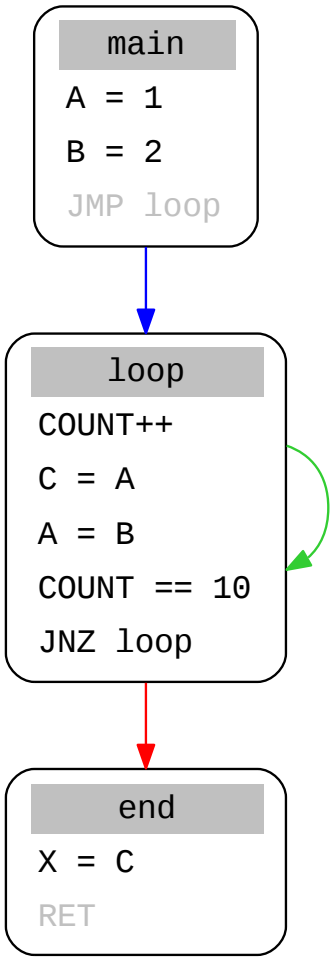


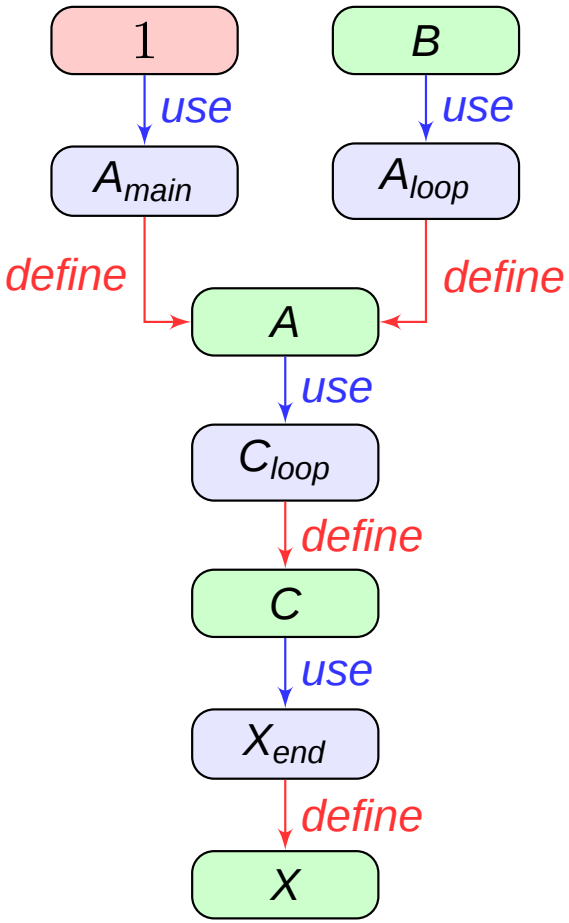
Use-define chains : Graphe de dépendances



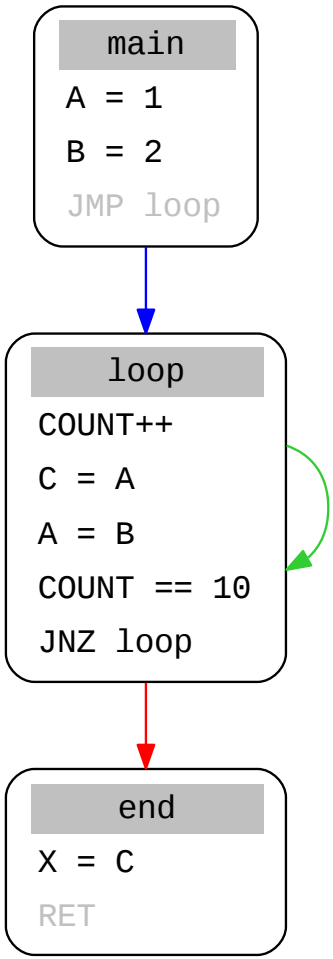


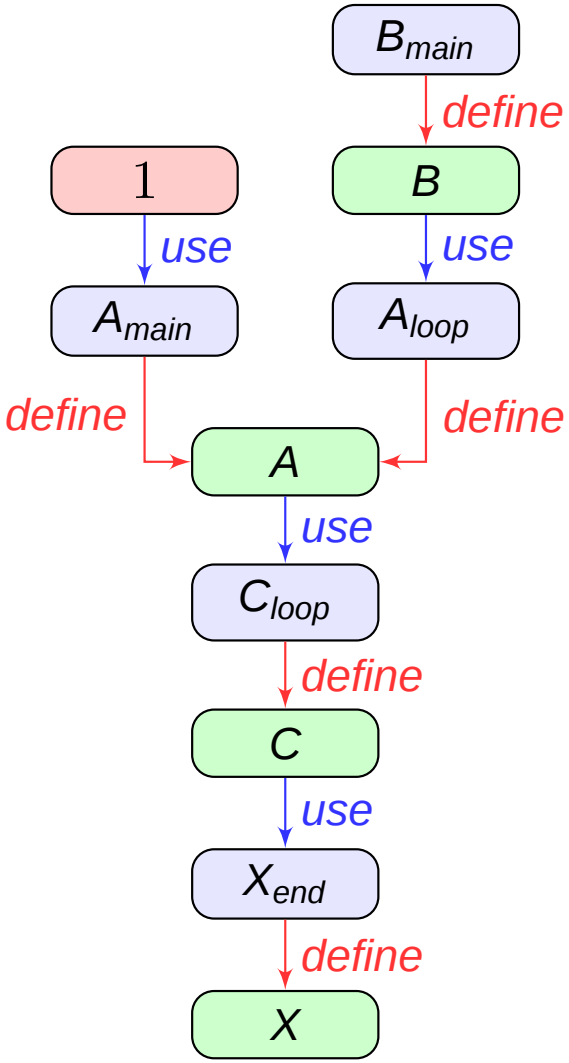
Use-define chains : Graphe de dépendances



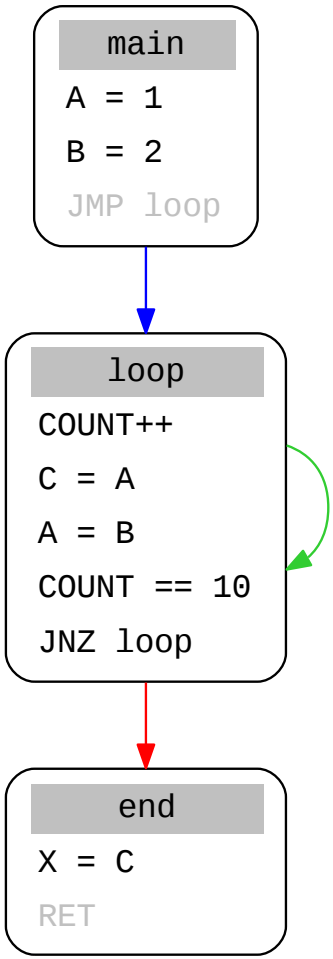


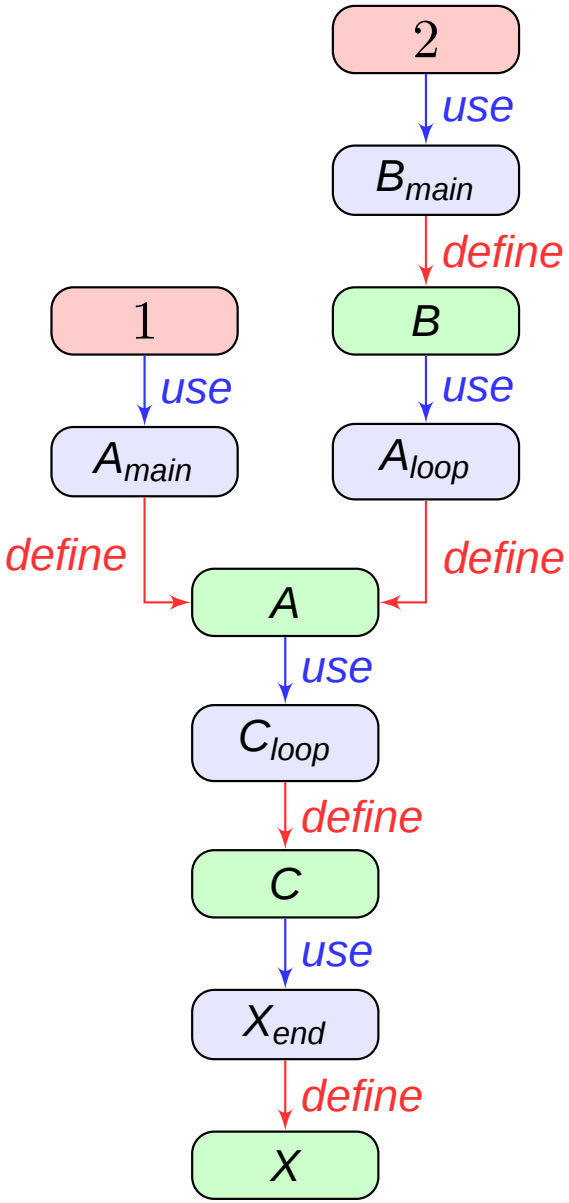
Use-define chains : Graphe de dépendances



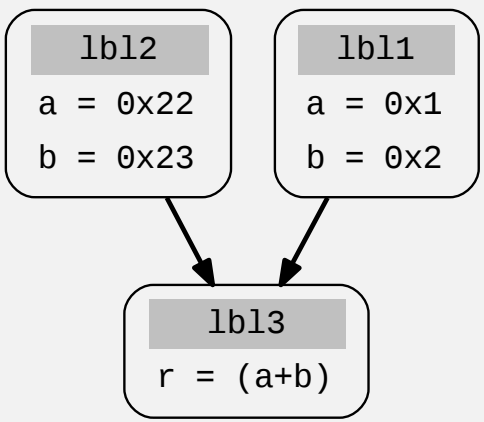


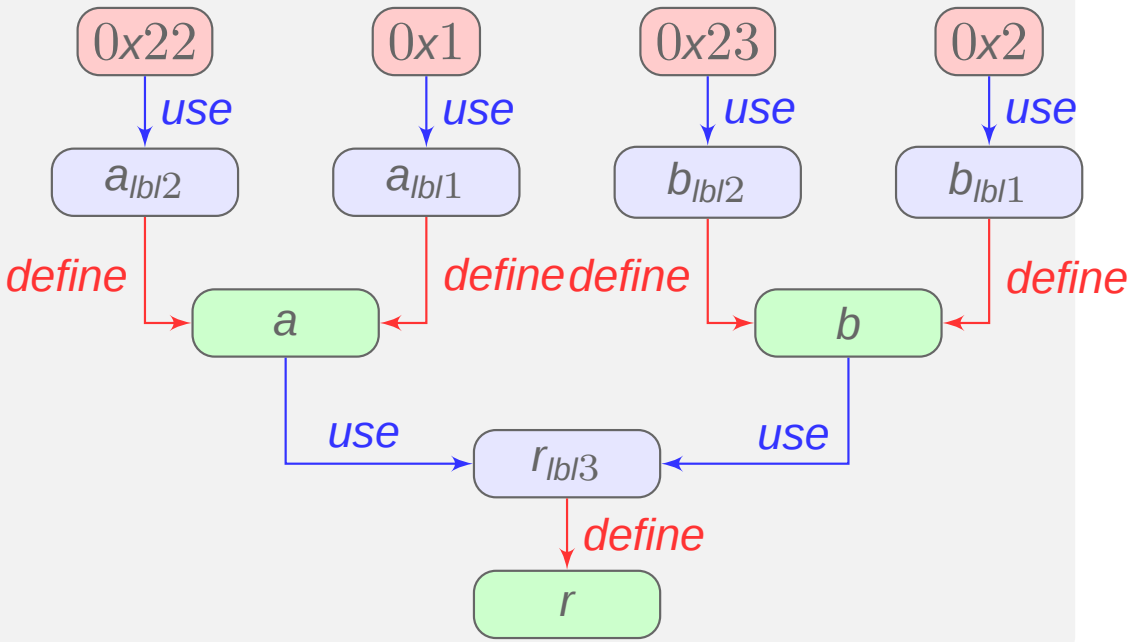
Use-define chains : Graphe de dépendances





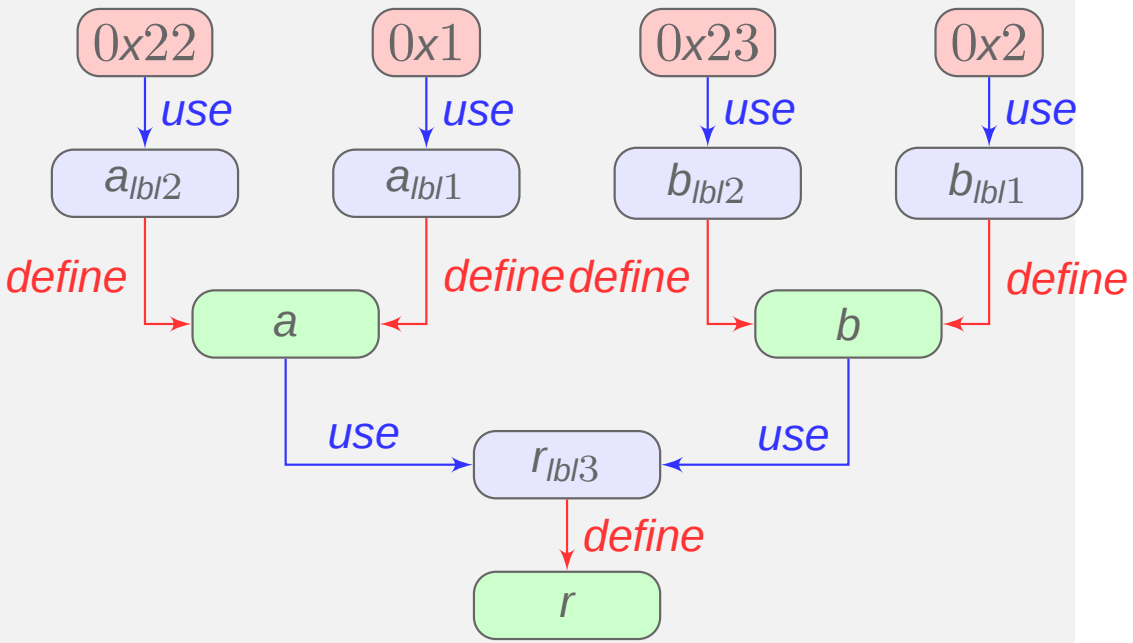
Inconvénient : sur-ensemble des solutions





Inconvénient : sur-ensemble des solutions

1. $a : 0x1, b : 0x2$
2. $a : 0x22, b : 0x23$
3. $a : 0x1, b : 0x23$
4. $a : 0x22, b : 0x2$



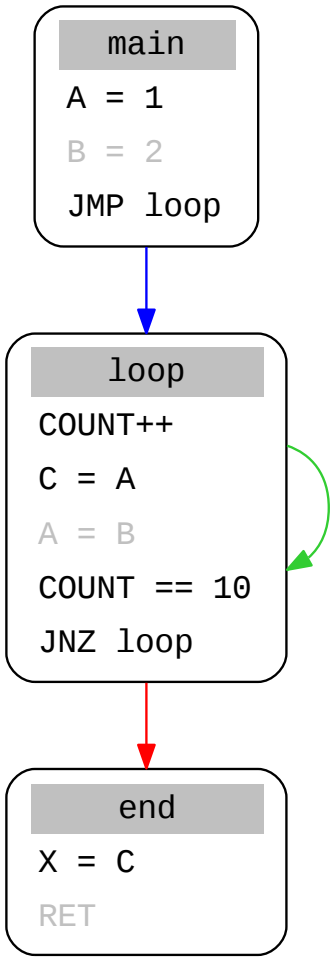
Angr : Path-sensitive

Path-sensitive

- Exécution symbolique
- Branchement $\longrightarrow$ contrainte
- “PathGroup”
- Magie noire pour les boucles

```
state = project.factory.blank_state(addr=0x401000)

# Activating veritesting results in a crash
path_group = project.factory.path_group(state, veritesting=False)
path_group = path_group.explore(find=(addr,))

# Display results
founds = path_group.found
for i, found in enumerate(founds):
   print i, found.state.regs.eax
```

Angr Path-sensitive




```
$ python angr_pathgroup.py

 - COUNT = 9
 - C = 1
```

Path-sensitive

Cas d'utilisations

- Recherche de vulnérabilités, crackme, …
- Participation au "Cyber Grand Challenge" (équipe *Shellphish*)

Inconvénients

- Nécessité d'une heuristique …
- …quitte à oublier des solutions
  - `COUNT != 9`
  - `C = 2`
- Très lourd
- Quid des boucles inutiles ?

Trouver un compromis

Marchand de cailloux

Solutions sur étagère disponibles :

- Contenant trop peu d'information (sac de sable)
- Trop lourde à utiliser (sac de menhirs)

Caractéristiques désirées

- Plus précis que l'algorithme des *use-define chains*
- Mais sans avoir peur des boucles

Trouver un compromis

Marchand de cailloux

Solutions sur étagère disponibles :

- Contenant trop peu d'information (sac de sable)
- Trop lourde à utiliser (sac de menhirs)

Caractéristiques désirées

- Plus précis que l'algorithme des *use-define chains*
- Mais sans avoir peur des boucles

Metasm : *BackTracking*

Metasm : `metasm.cr0.org`

- “Cross-architecture assembler, disassembler, compiler, linker and debugger”
- Loader (PE, ELF, Mach-O)
- Multi-architecture (arc, arm, arm64, …, st20, x86 64, z80)

BackTracking

Méthode : un seul passage dans chaque boucle

```
pe = PE.decode_file(filename)
dasm = pe.disassemble_fast_deep ’entrypoint’
reg = reg.to_sym

# Run backtrace
log = []
dasm.debug_backtrace = 1
ret = dasm.backtrace(reg, offset, :log => log)
```

Metasm

Metasm

- “use-define chains”  “path-sensitive”
- Analyse très rapide
- Trouve la solution `x = 1`
- Indique qu’il a potentiellement manqué des solutions

Raffinement de la solution

On pense pouvoir obtenir mieux pour pas beaucoup plus cher.

I have a dream

Avantages de l'algorithme présenté (vue du commercial)

- Distinguer les chemins d'exécutions
- Éviter les chemins "équivalents en dépendance"
- Dérouler les boucles seulement le nombre de fois nécessaire

$\rightarrow$ un sac de *pocket*-cailloux !

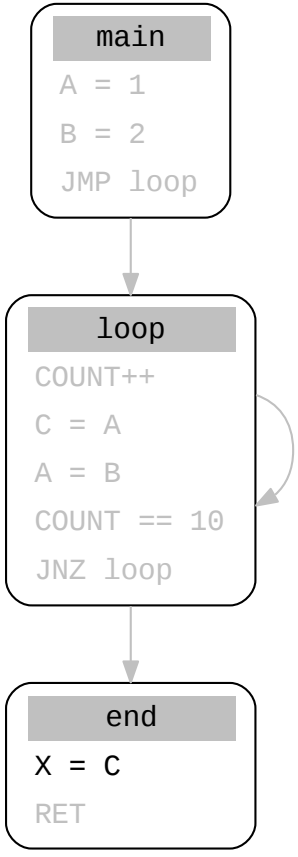
Algorithme

Étapes

1. L’algorithme suit les dépendances dans le *basic block* courant
2. On poursuit l’analyse dans chaque bloc parent
3. Sauf ceux qu’on a déjà parcouru avec des dépendances identiques
4. L’algorithme s’arrête si on atteint la racine du graphe, ou que toutes les dépendances sont résolues.

Étapes

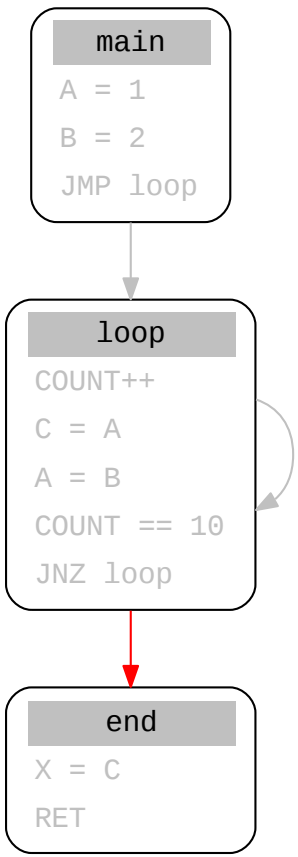
L'algorithme suit les dépendances dans le *basic block* courant



x dépend de c

Étapes

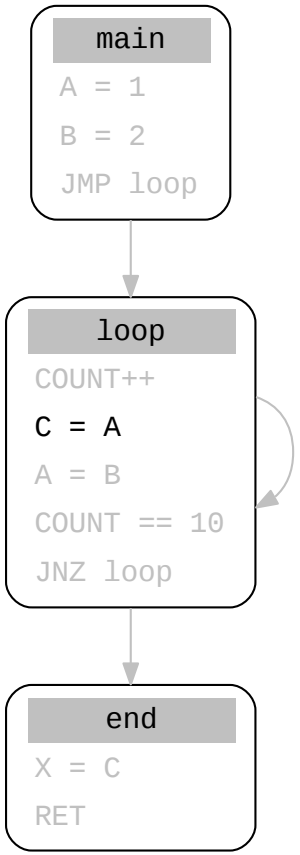
On poursuit l'analyse dans chaque bloc parent



De qui dépend `c` dans `loop` ?

Étapes

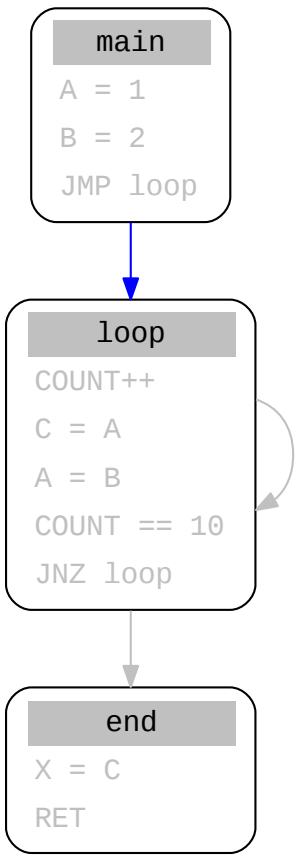
L'algorithme suit les dépendances dans le *basic block* courant



C dépend de A

Étapes

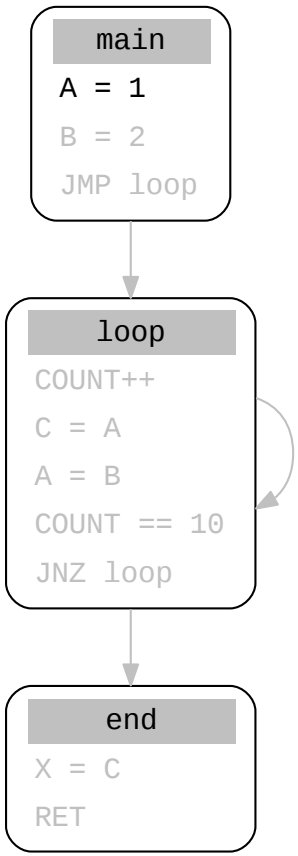
On poursuit l'analyse dans chaque bloc parent



De qui dépend `A` dans `main` ?

Étapes

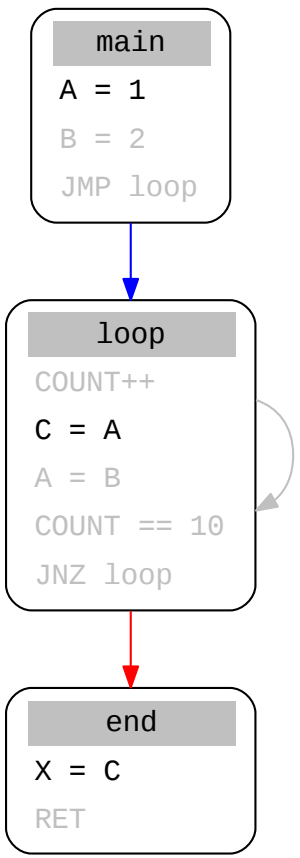
L'algorithme suit les dépendances dans le *basic block* courant



A dépend de la constante 1

Étapes

L'algorithme s'arrête si toutes les dépendances sont résolues.

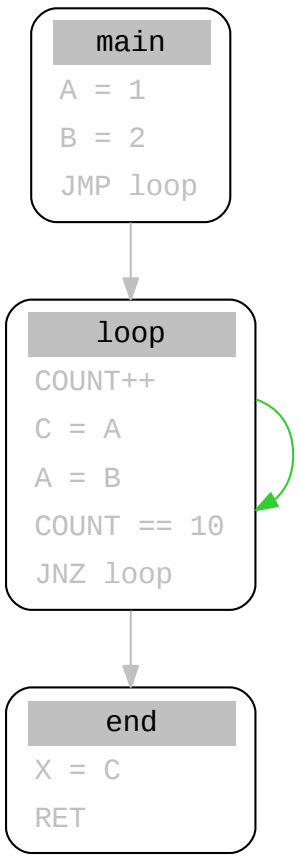


Solution 1 :

1. $X_{fin\ end} \leftarrow C_{debut\ end}$
2. $C_{fin\ loop} \leftarrow A_{debut\ loop}$
3. $A_{fin\ main} \leftarrow 1$

Étapes

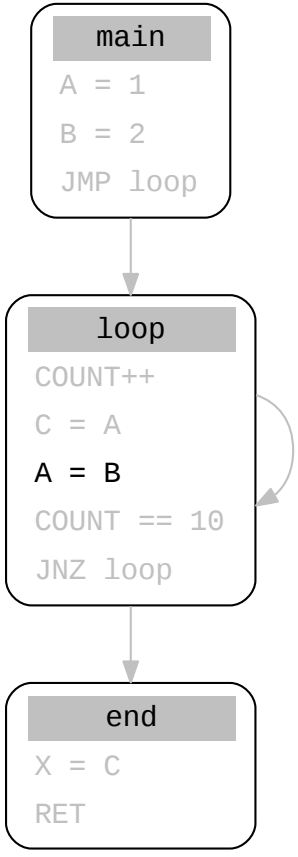
On poursuit l'analyse dans chaque bloc parent



De qui dépend `A` dans `loop` ?

Étapes

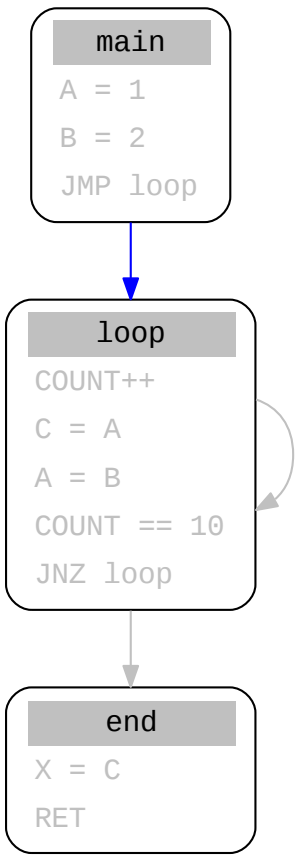
L'algorithme suit les dépendances dans le *basic block* courant



A dépend de B

Étapes

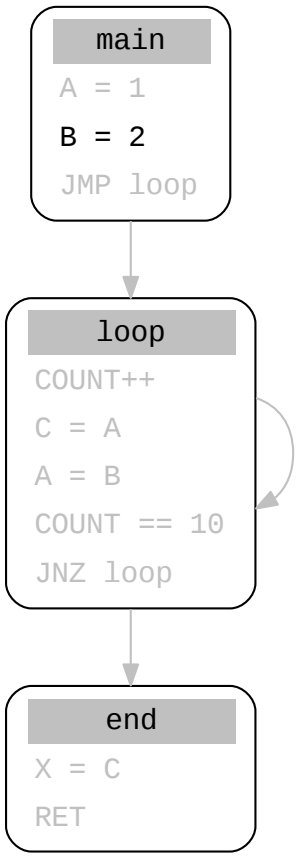
On poursuit l'analyse dans chaque bloc parent



De qui dépend `B` dans `main` ?

Étapes

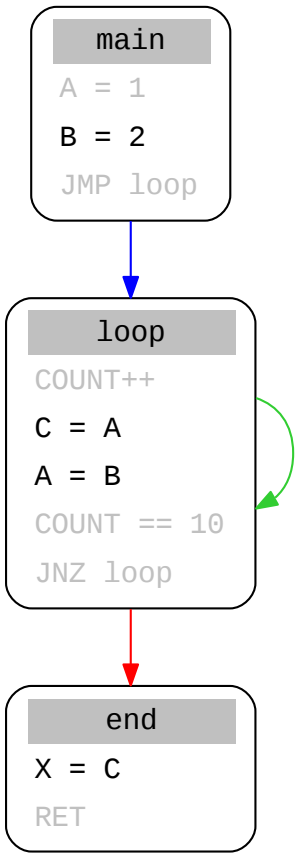
L'algorithme suit les dépendances dans le *basic block* courant



`B` dépend de la constante `2`

Étapes

L'algorithme s'arrête si toutes les dépendances sont résolues.

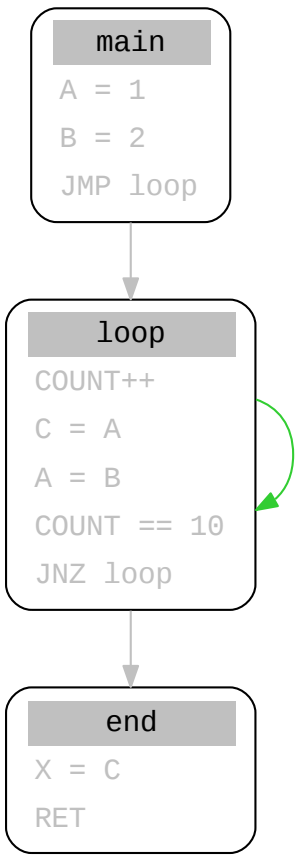


Solution 2 :

1. $X_{fin\ end} \leftarrow C_{debut\ end}$
2. $C_{fin\ loop} \leftarrow A_{debut\ loop}$
3. $A_{fin\ loop} \leftarrow B_{debut\ loop}$
4. $B_{fin\ main} \leftarrow 2$

Étapes

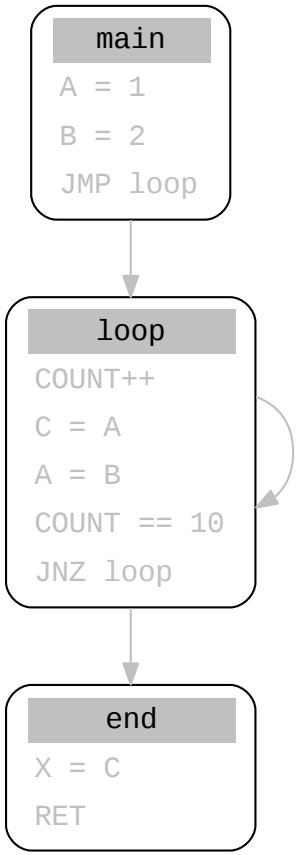
On poursuit l'analyse dans chaque bloc parent



De qui dépend `B` dans `loop` ?

Étapes

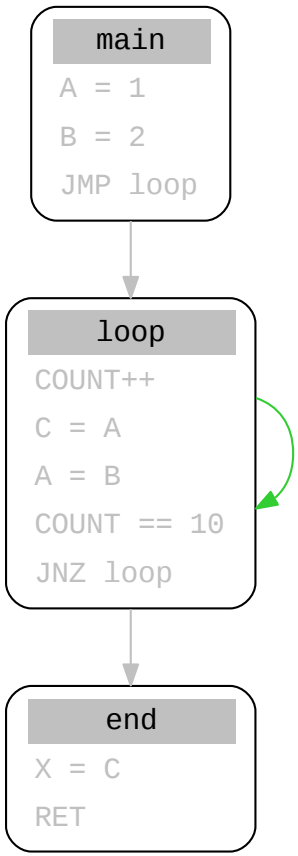
L'algorithme suit les dépendances dans le *basic block* courant



`B` n'est pas modifié

Étapes

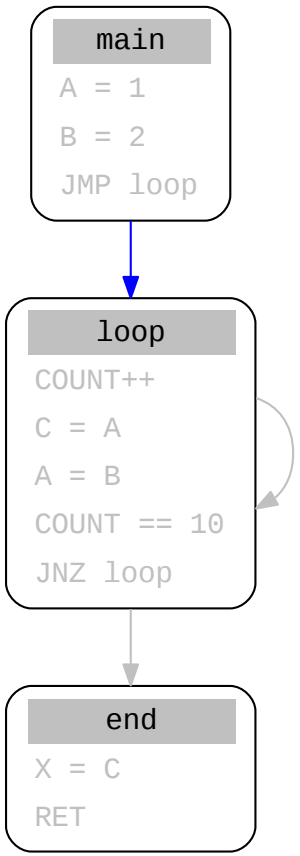
Sauf les parents qu'on a déjà parcourus avec des dépendances identiques



On arrête cette branche de recherche

Étapes

Sauf les parents qu'on a déjà parcourus avec des dépendances identiques



On arrête cette branche de recherche

Algorithme

Étapes

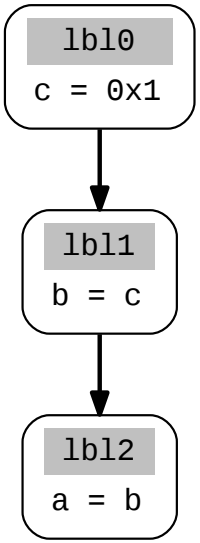
L'algorithme s'arrête si on atteint la racine du graphe, ou que toutes les dépendances sont résolues.

Fin de l'algorithme

Il n'y a plus de dépendances à résoudre. Deux solutions ont été trouvées :

- `X = 1` avec **un seul** passage dans `loop`
- `X = 2` avec **plus d'un** passage dans `loop`

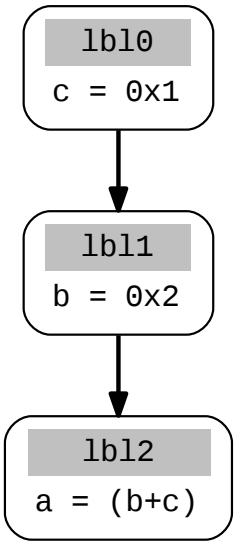
Résultat



Solution

1. $a \leftarrow b_{lbl2}$
2. $b_{lbl2} \leftarrow c_{lbl1}$
3. $c_{lbl1} \leftarrow 0x1_{lbl0}$

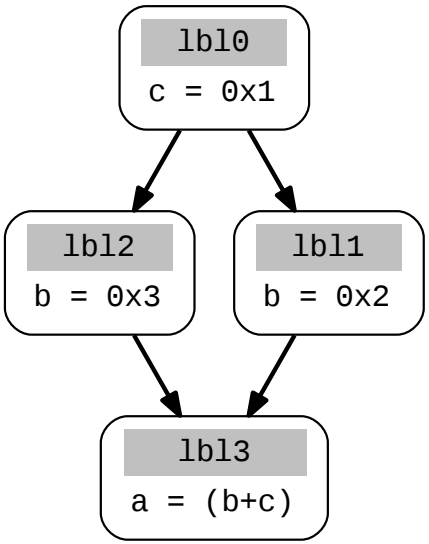
Résultat



Solution

1. $a \leftarrow b_{lbl2}, c_{lbl2}$
2. $b_{lbl2} \leftarrow 0x2_{lbl1}$
3. $c_{lbl2} \leftarrow 0x1_{lbl0}$

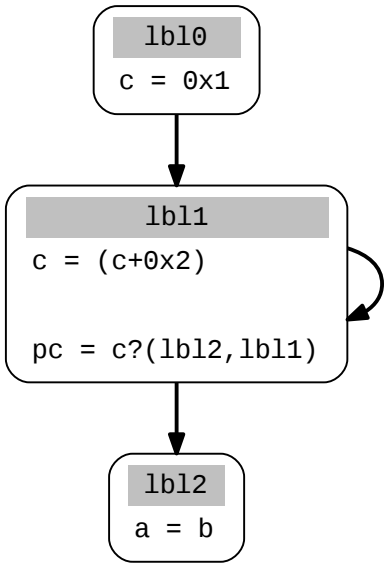
Résultat



Solutions

1 $a \leftarrow b_{lbl3}, c_{lbl3}$	1 $a \leftarrow b_{lbl3}, c_{lbl3}$
2 $b_{lbl3} \leftarrow 0x2_{lbl1}$	2 $b_{lbl3} \leftarrow 0x3_{lbl2}$
3 $c_{lbl3} \leftarrow 0x1_{lbl0}$	3 $c_{lbl3} \leftarrow 0x1_{lbl0}$

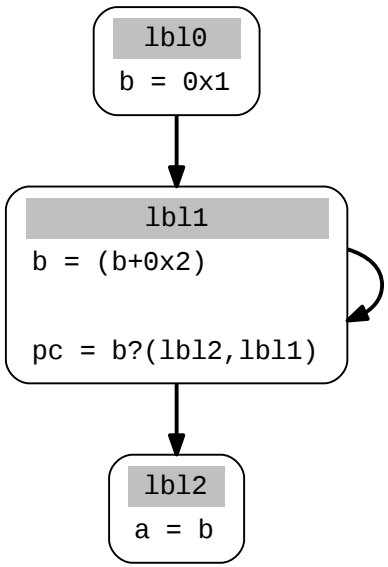
Résultat



Solution

1. $a \longleftarrow b_{lbl2}$
2. $b_{lbl2} \longleftarrow b_{head}$

Résultat



Solutions

1. $a \leftarrow b_{lbl2}$
2. $b_{lbl2} \leftarrow b_{lbl1}$
3. $b_{lbl1} \leftarrow 0x1_{lbl0}$

1. $a \leftarrow b_{lbl2}$
2. $b_{lbl2} \leftarrow b_{lbl1}$
3. $b_{lbl1} \leftarrow b_{lbl1}$
4. $b_{lbl1} \leftarrow 0x1_{lbl0}$

Partie formelle

Terminaison de l'algorithme

Voir les actes. Spoiler : il termine

Complexité

- La complexité s'obtient directement en connaissant le nombre de solutions …
- qui s'obtient en travaillant sur un "algorithme adapté" …
- qui se base sur un "CFG adapté" …
- qui est détaillé dans les actes
- $\rightarrow 2^{|v|^2+|x|} + |v|.2^{|v|^2}$

Majorant du nombre de solution

- 16 blocs : nombre d'atomes dans l'univers
- 32 blocs : personnel de l'ANSSI

Partie formelle

Terminaison de l'algorithme

Voir les actes. Spoiler : il termine

Complexité

- La complexité s'obtient directement en connaissant le nombre de solutions …
- qui s'obtient en travaillant sur un "algorithme adapté" …
- qui se base sur un "CFG adapté" …
- qui est détaillé dans les actes
- $\rightarrow 2^{|V|^2+|X|} + |V|.2^{|V|^2}$

Majorant du nombre de solution

- 16 blocs : nombre d'atomes dans l'univers
- 32 blocs : personnel de l'ANSSI

Partie formelle

Terminaison de l'algorithme

Voir les actes. Spoiler : il termine

Complexité

- La complexité s'obtient directement en connaissant le nombre de solutions …
- qui s'obtient en travaillant sur un "algorithme adapté" …
- qui se base sur un "CFG adapté" …
- qui est détaillé dans les actes
- $\rightarrow 2^{|v|^2+|x|} + |v|.2^{|v|^2}$

Majorant du nombre de solution

- 16 blocs : nombre d'atomes dans l'univers
- 32 blocs : personnel de l'ANSSI

Résultat : analyse du *Memcpy*

```
2 solutions:
Possible value: @32[@32[ESP_init+0x4]+0x10] - @32[ESP_init+0x8]
Possible value: @32[ESP_init+0xC]
```

Résultat : analyse du *duqu2*

Recherche de la source des arguments

Fonction : 0x10003925

Sample :

52fe506928b0262f10de31e783af8540b6a0b232b15749d647847488acd0e17a

```
DISAMS
INFO : generating IR... 10003959
Function: 0x10003925L Reference: 0x10003959L
        Solution: 0 has_loop: False Arguments values: ['0x10032324', '0xC9052BB8']
Function: 0x10003925L Reference: 0x10003968L
        Solution: 0 has_loop: False Arguments values: ['0x10032324', '0xEAF3865C']
Function: 0x10003925L Reference: 0x1000397fL
        Solution: 0 has_loop: False Arguments values: ['0x10032324', '0xAF32FB74']
Function: 0x10003925L Reference: 0x1000398eL
        Solution: 0 has_loop: False Arguments values: ['0x10032324', '0xC9EE1C94']
Function: 0x10003925L Reference: 0x1000399dL
...
```

Limitations : résolution des alias

```
mov @32[B], 0x6
mov @32[C], 0x18
mov A, @32[B]
```

Cas possibles

- Les cases mémoire `[B]` et `[C]` sont disjointes, A $= 0x6$
- B et C ont la même valeur, A $= 0x18$
- Les cases mémoire se recouvrent partiellement, A aura une autre valeur

Limitations : pointeur de pile non résolu

Langage natif	Langage intermédiaire
`push edx`	`ESP = ESP - 0x4` `@32[ESP] = EDX`
`push [ebp+arg_4]`	`ESP = ESP - 0x4` `@32[ESP] = @32[EBP + 0xC]`

`call 0x10028BA2`	`ESP = ESP - 0x4`
	`EIP = 0x10028BA2`
	`@32[ESP] = 0x10028B84`
	`IRDst = lbl_gen_00000000`

Limitations : pointeur de pile non résolu

Langage natif	Langage modifié
`push edx`	`ESP = ESP_init - 0xC` `@32[ESP_init - 0xC] = EDX`
`push [ebp+arg_4]`	`ESP = ESP_init - 0x10` `@32[ESP_init - 0x10] = @32[EBP + 0xC]`

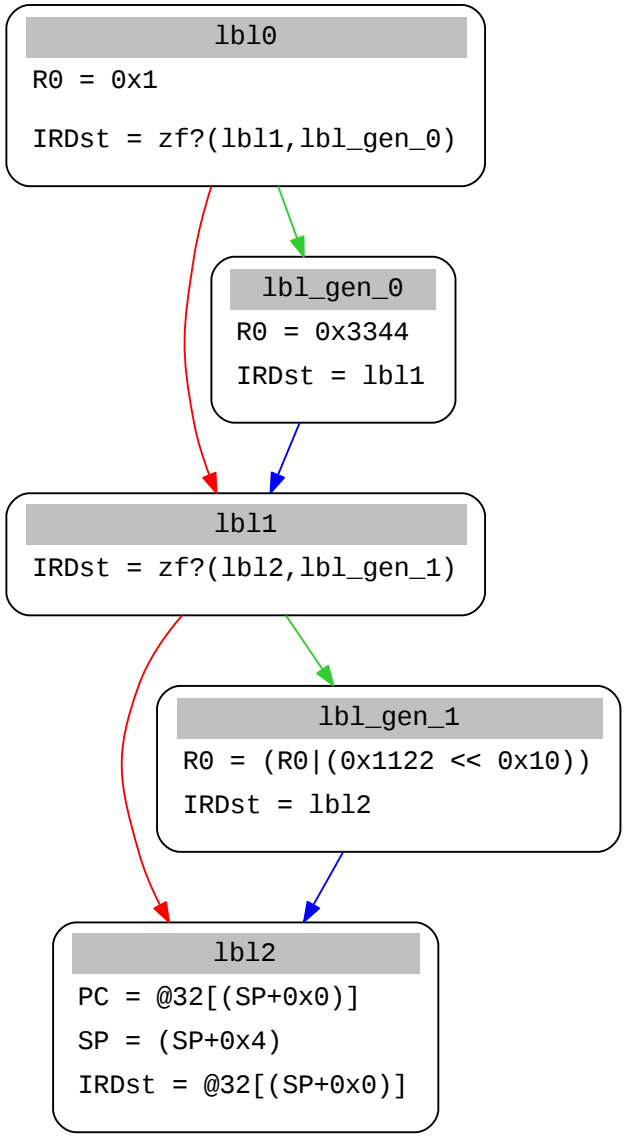
`call 0x10028BA2`	`ESP = ESP_init - 0x14`
	`EIP = 0x10028BA2`
	`@32[ESP_init - 0x14] = 0x10028B84`
	`IRDst = lbl_gen_00000000`

Si on ne suit pas les bonnes dépendances …

Limitations : chemins non satisfaisables

```
if (COND)
    R0 = 1;
else
    R0 = 0x11223344;
```

```
MOV     R0, 0x1
MOVNE   R0, 0x3344
MOVTNE  R0, 0x1122
```



Le Petit Poucet se dote du DepGraph

Conclusion : Miasm, l’ami des tout petits

Veni, Vidi, Vidi

- Mix entre les deux familles d’analyses
- Limitations contournables en pratique
- Explicite/Implicite

Merci !

http://www.miasm.re/blog

@MiasmRE

Commissariat à l’énergie atomique et aux énergies alternatives CEA/DAM
Centre de Bruyères-le-Châtel │ 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 │ F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019

Partie formelle

Une famille de CFG ayant de nombreuses solutions

$$2^{|V|log(|V|)} < solutions < 2^{|V|^2+|x|} + |V|.2^{|V|^2}$$