The Penultimate Challenge:
Bug report construction in the Clang Static Analyzer

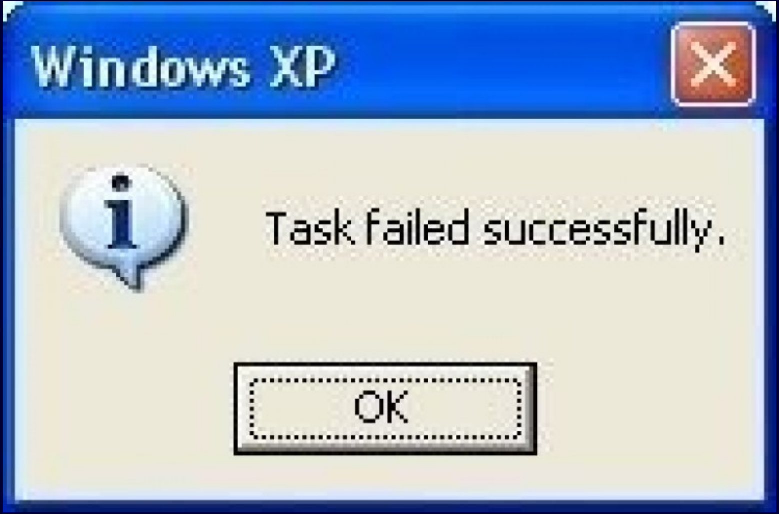
Kristóf Umann

dkszelethus@gmail.com

Eötvös Loránd University, Budapest

Ericsson Hungary

2019. oct. 22.



Task failed successfully.

Clear, precise bug reports are important

- One of the main selling points of Clang back in the day
- Not only wording, it requires a good infrastructure
- Tools without it are miserable to use

Agenda

- Path-sensitive analysis in the Clang Static Analyzer
- Current state of bug report construction
- Difficulties, current state of research, future work

Path-sensitive analysis in the Clang Static Analyzer

The Clang Static Analyzer

It employs a variety of techniques to analyze C, C++, ObjectiveC, ObjectiveC++ code:

- AST matching
- CFG based analyses
- Symbolic execution

Exploring paths of execution

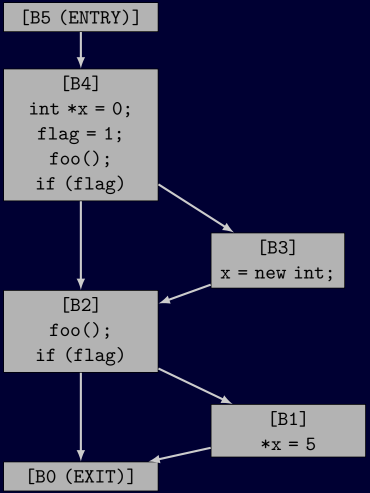
- Traverse the control flow graph (CFG) of a function
- On branches, explore a path on which the condition is true, and one on which its false
- How does this work interprocedurally?

Exploring paths of execution

- Traverse the control flow graph (CFG) of a function
- On branches, explore a path on which the condition is true, and one on which its false
- How does this work interprocedurally? Inlining!

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

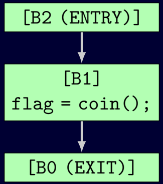
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:

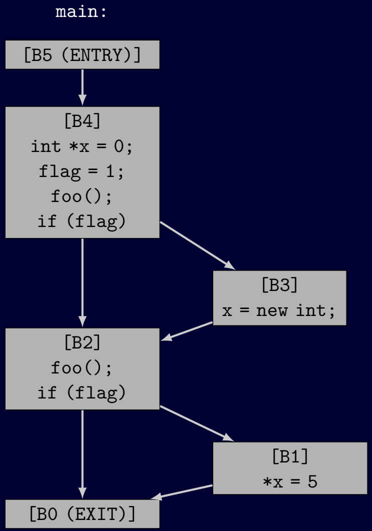


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

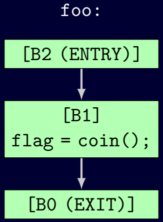
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



foo:

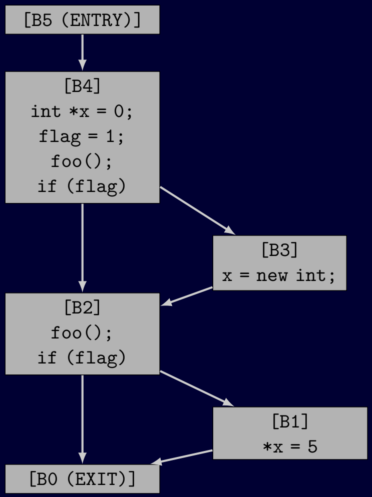
[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

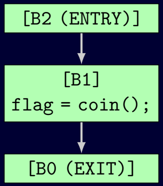
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:



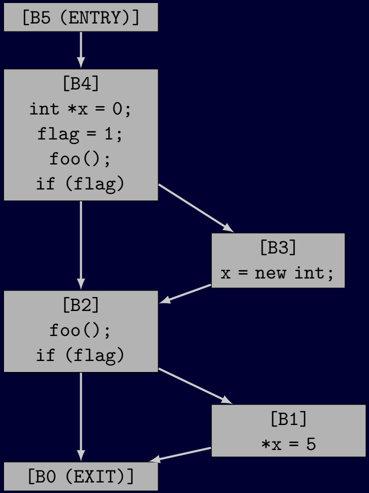
[B2 (ENTRY)]

[B1]
flag = coin();

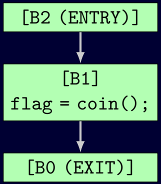
[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:

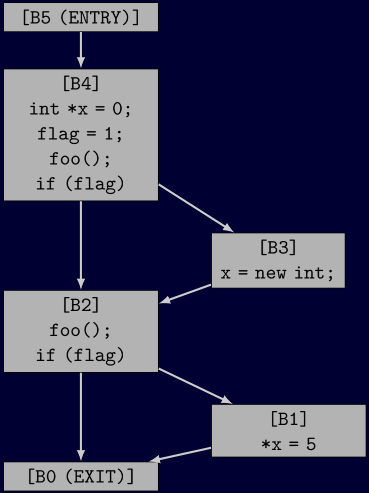


foo:



```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

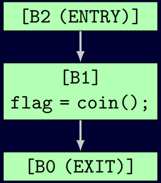
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:

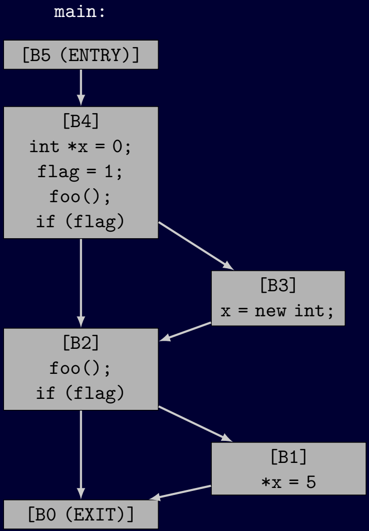


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

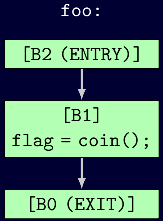
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



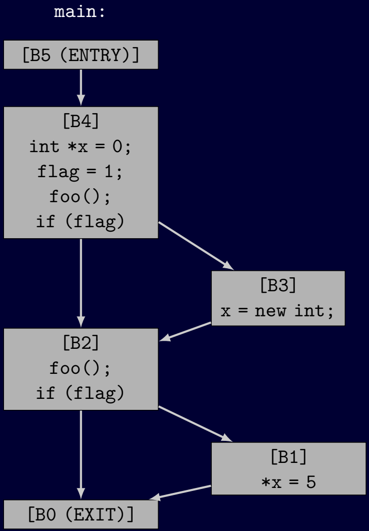
foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

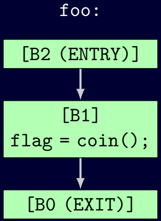
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



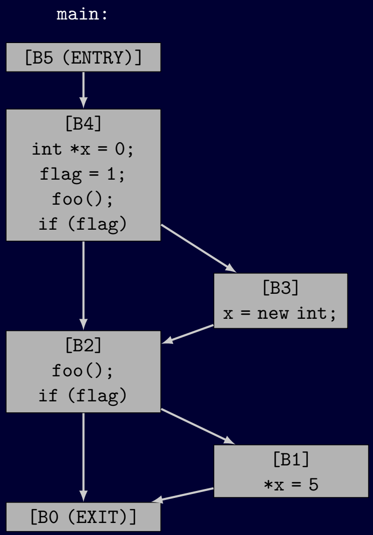
foo:

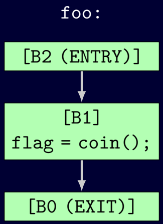
[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

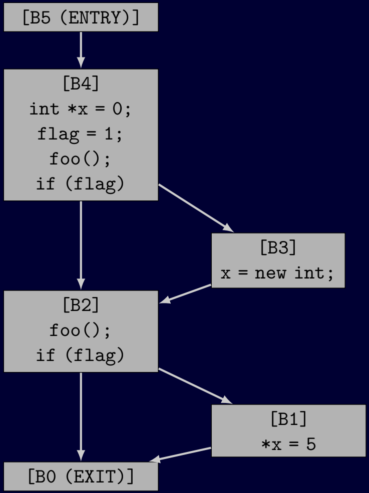
```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



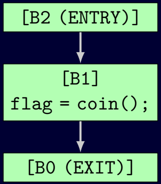


```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:

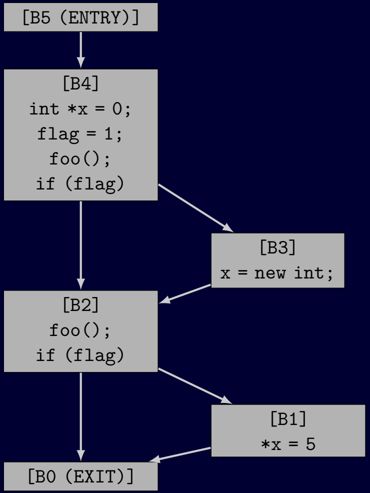


foo:



```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

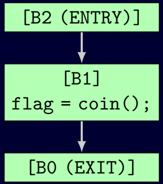
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:

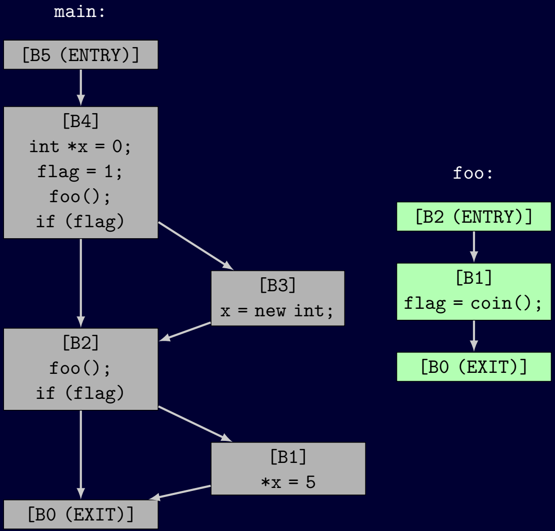


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

```
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
```

```
[B3]
x = new int;
```

```
[B2]
foo();
if (flag)
```

```
[B1]
*x = 5
```

[B0 (EXIT)]

foo:

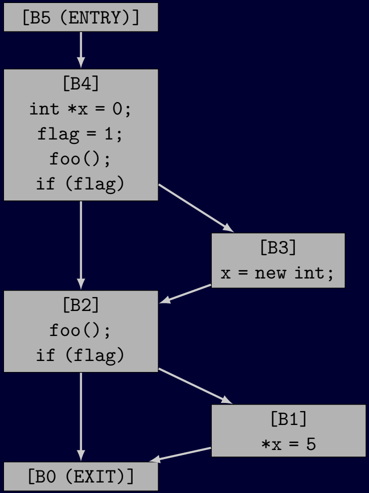
[B2 (ENTRY)]

```
[B1]
flag = coin();
```

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

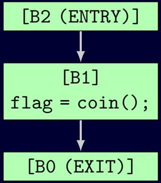
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

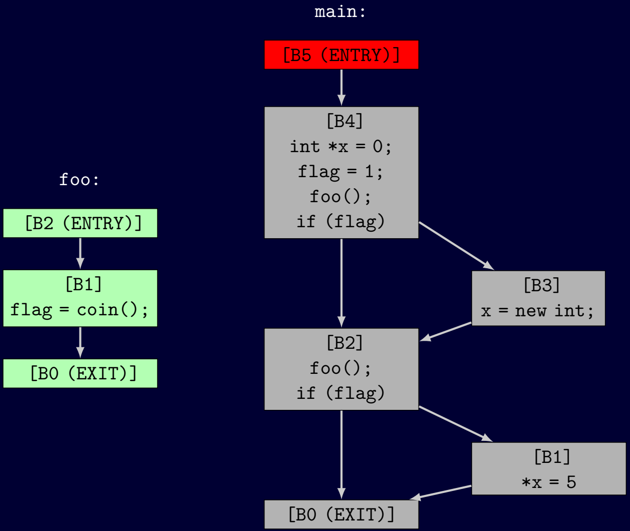
foo:



[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

```
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
```

```
[B3]
x = new int;
```

```
[B2]
foo();
if (flag)
```

```
[B1]
*x = 5
```

[B0 (EXIT)]

foo:

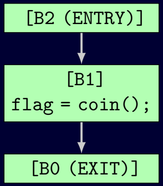
[B2 (ENTRY)]

```
[B1]
flag = coin();
```

[B0 (EXIT)]

```
flag == 1;
x == nullptr;
```

foo:

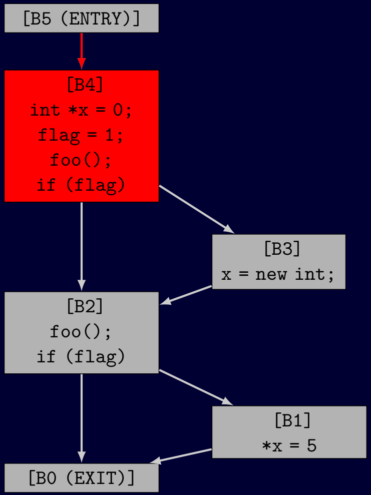


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

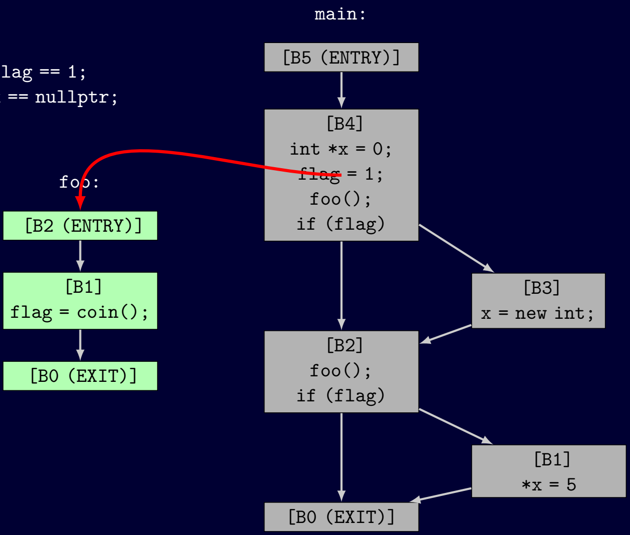
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

```
flag == 1;
x == nullptr;
```



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:

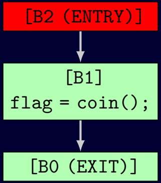
[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

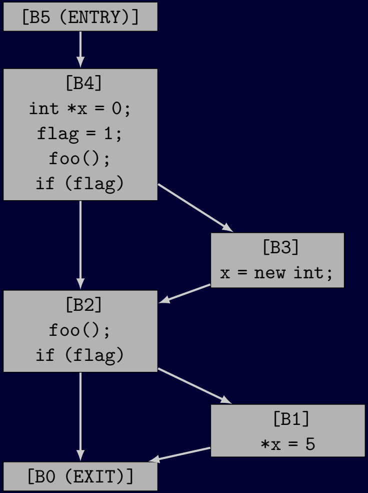
```
flag == 1;
x == nullptr;
```

foo:



```
[B2 (ENTRY)]
[B1]
flag = coin();
[B0 (EXIT)]
```

main:

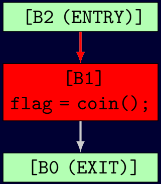


```
[B5 (ENTRY)]
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
[B3]
x = new int;
[B2]
foo();
if (flag)
[B1]
*x = 5
[B0 (EXIT)]
```

main:

flag $\in (-\infty, \infty)$;
x == nullptr;

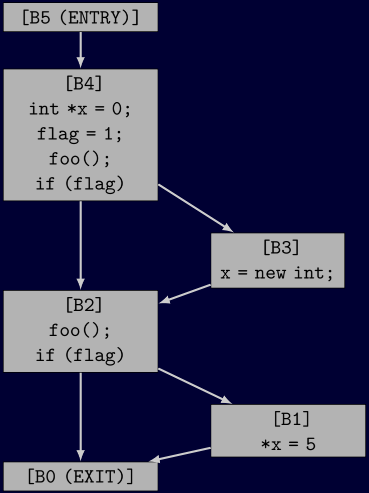
foo:



[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

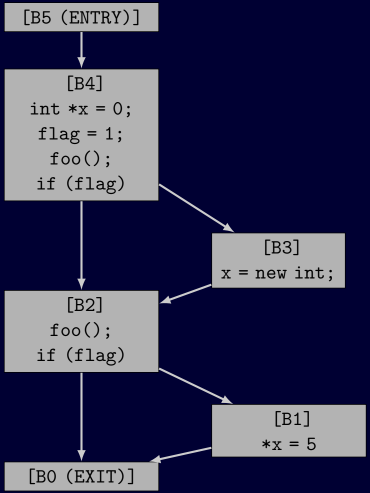
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

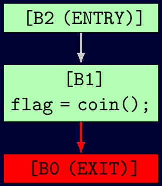
[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag $\in (-\infty, \infty)$;
x == nullptr;

foo:

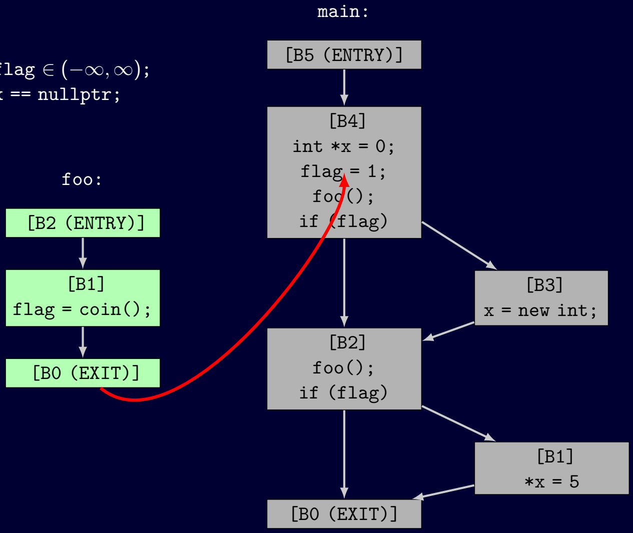


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

flag ∈ $(-\infty, \infty)$;
x == nullptr;



foo:

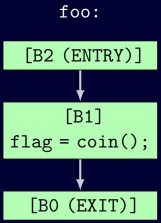
```
[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]
```

main:

```
[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]
```

$\texttt{flag} \in (-\infty, \infty)$;

x == nullptr;

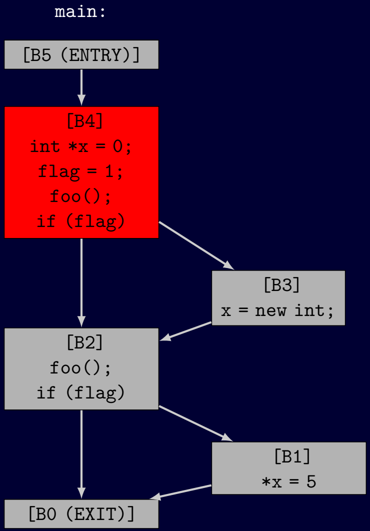


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

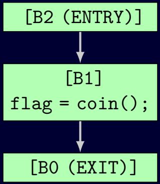
[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

```
flag == 0;
x == nullptr;
```

foo:

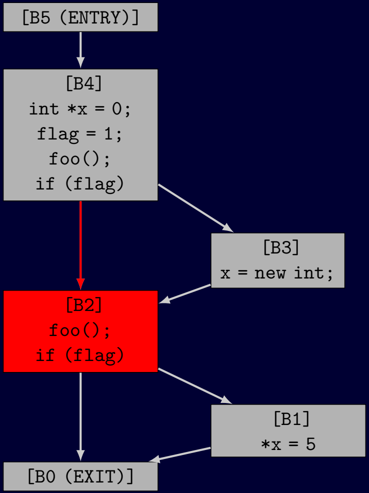


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

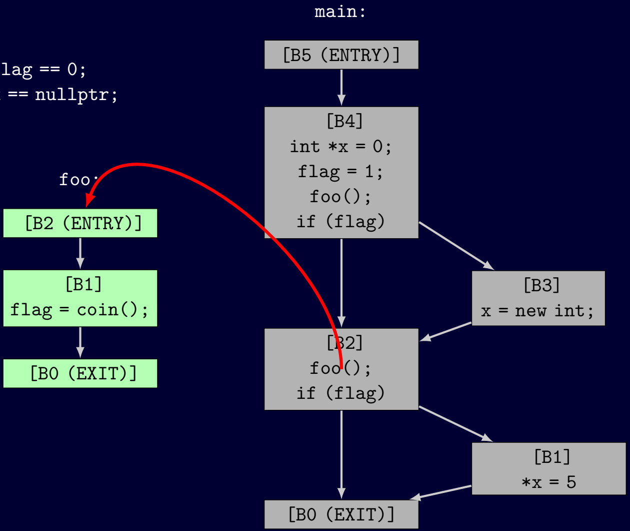
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

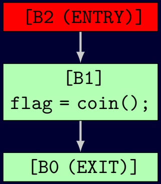
[B0 (EXIT)]

flag == 0;
x == nullptr;



```
flag == 0;
x == nullptr;
```

foo:

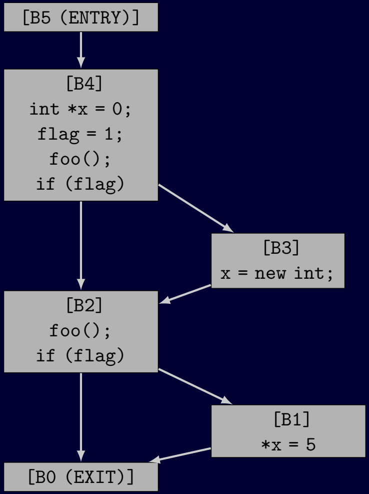


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

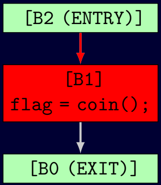
[B1]
*x = 5

[B0 (EXIT)]

$\texttt{flag} \in (-\infty, \infty)$;

x == nullptr;

foo:

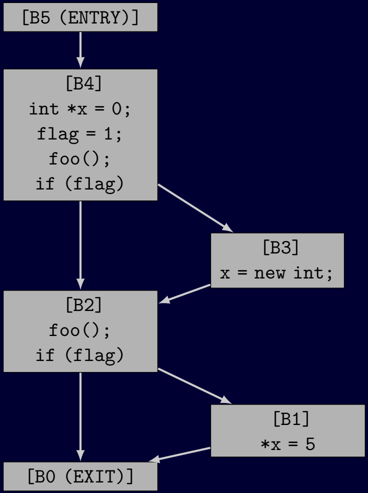


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

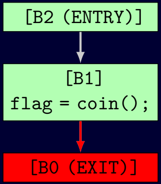
[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag $\in (-\infty, \infty)$;
x == nullptr;

foo:

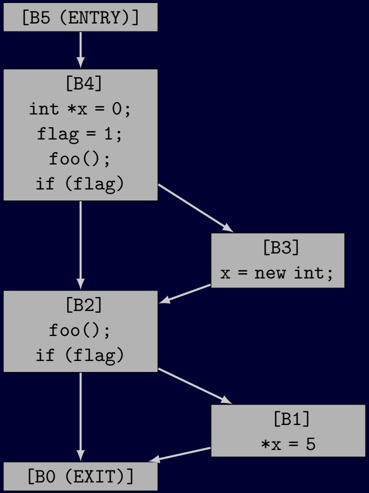


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

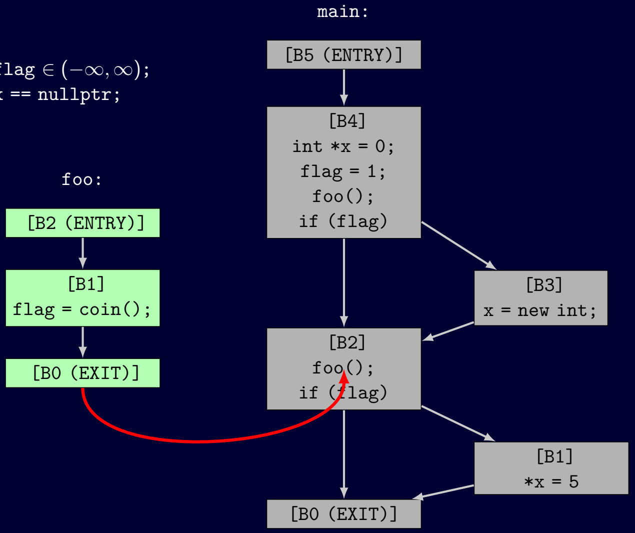
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

$\texttt{flag} \in (-\infty, \infty)$;
x == nullptr;



main:

[B5 (ENTRY)]

```
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
```

```
[B3]
x = new int;
```

```
[B2]
foo();
if (flag)
```

```
[B1]
*x = 5
```

[B0 (EXIT)]

foo:

[B2 (ENTRY)]

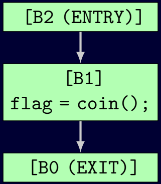
```
[B1]
flag = coin();
```

[B0 (EXIT)]

main:

flag $\in (-\infty, \infty)$;
x == nullptr;

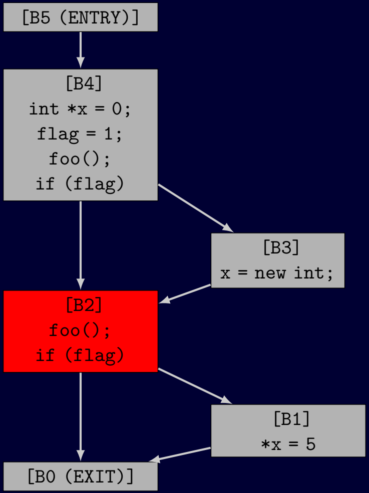
foo:



[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

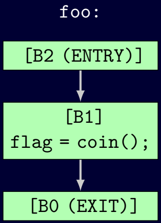
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag $\in (-\infty, 0) \cup (0, \infty)$;
x == nullptr;
dereference of x!

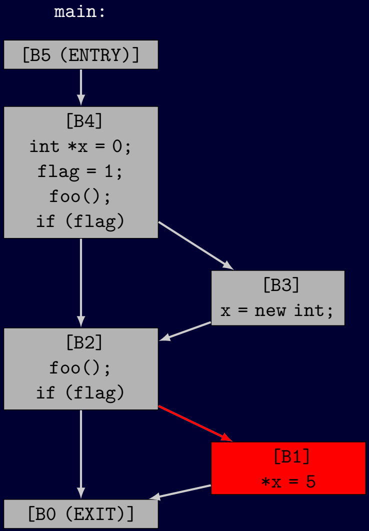


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

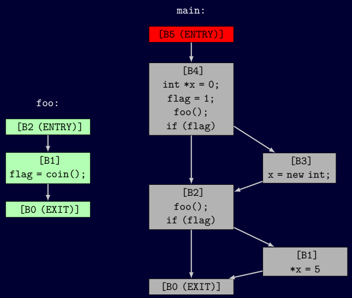
[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

The ExplodedGraph

- Contains everything the analyzer learned during symbolic execution
- All explored paths of execution
- Every symbolic value in every program state



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

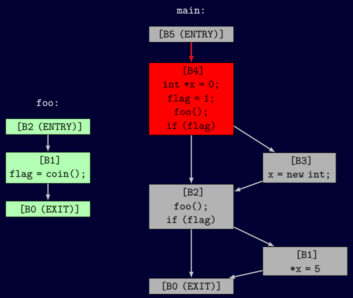
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

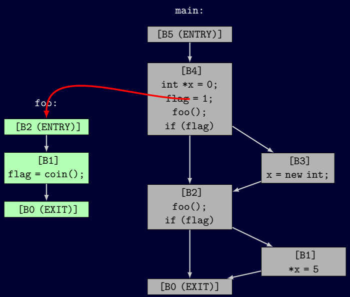
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

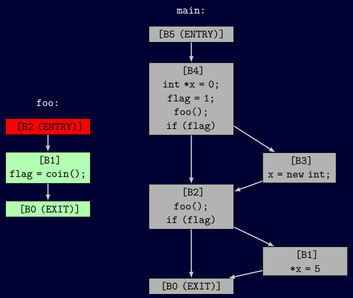
foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

flag = 1
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

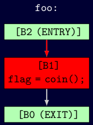
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

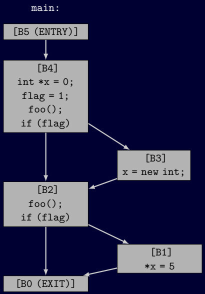


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

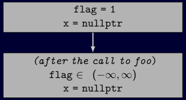
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

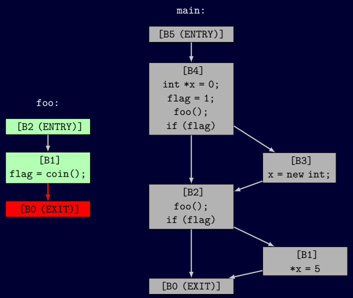
[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

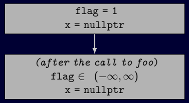
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

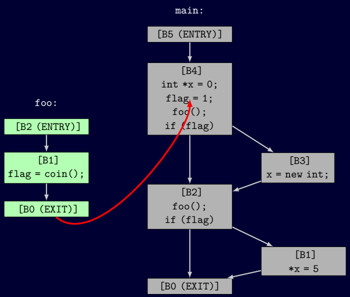
[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

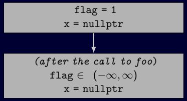
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

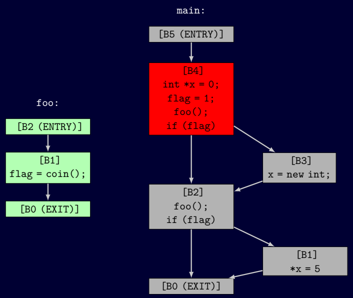
[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

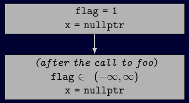
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

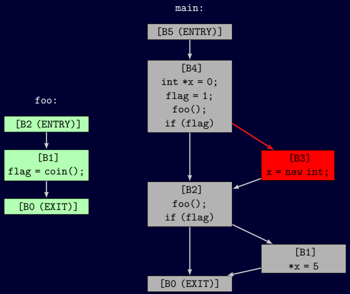
[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

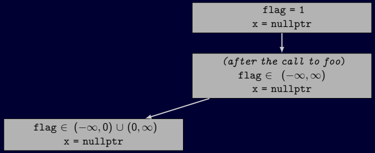
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

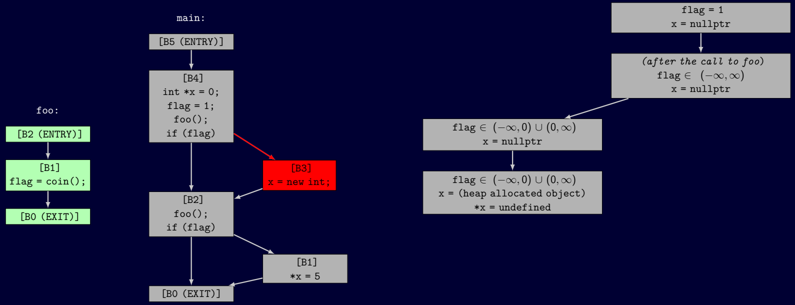
[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

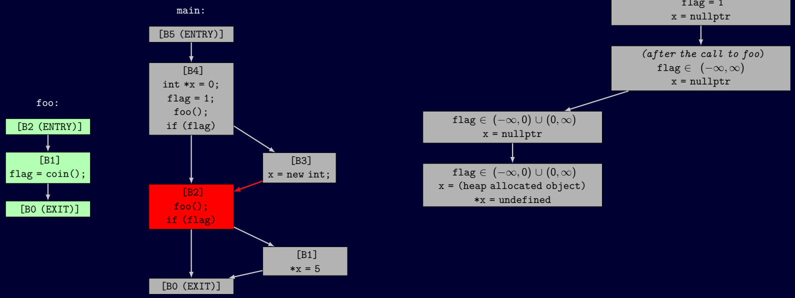
[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

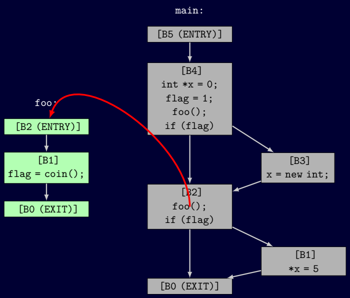
[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

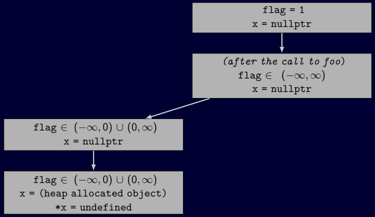
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

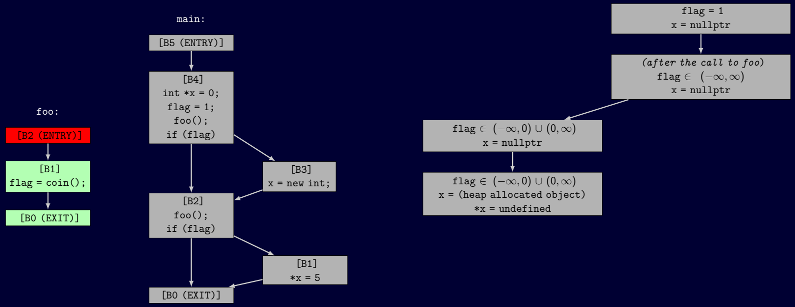


flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

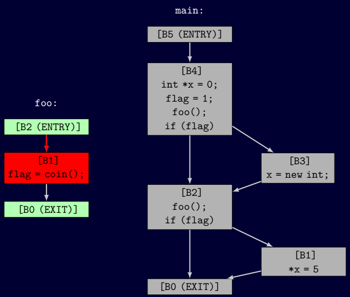
[B0 (EXIT)]

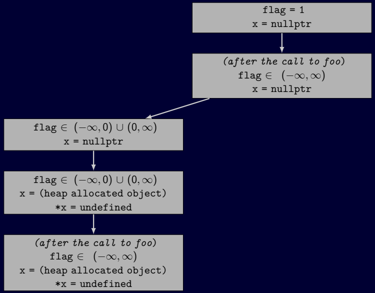
flag = 1
x = nullptr

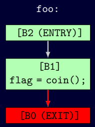
(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined


foo:
[B2 (ENTRY)]
[B1]
flag = coin();
[B0 (EXIT)]
main:
[B5 (ENTRY)]
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
[B3]
x = new int;
[B2]
foo();
if (flag)
[B1]
*x = 5
[B0 (EXIT)]


flag = 1
x = nullptr
(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined
(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

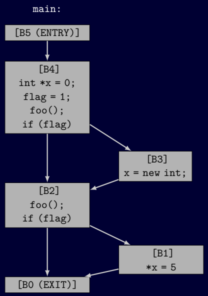


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

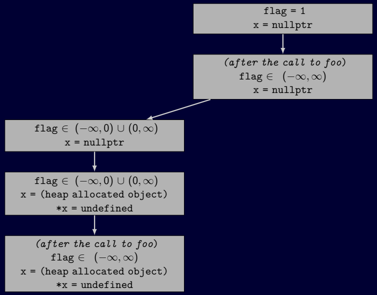
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



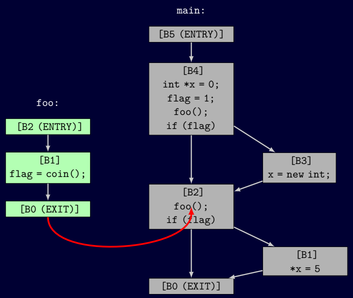
flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

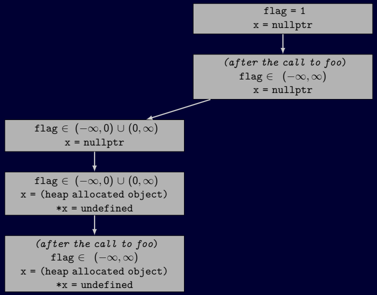
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



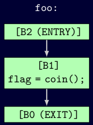
flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

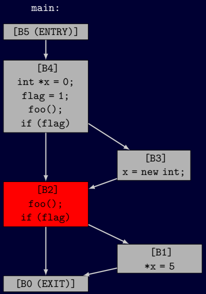


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

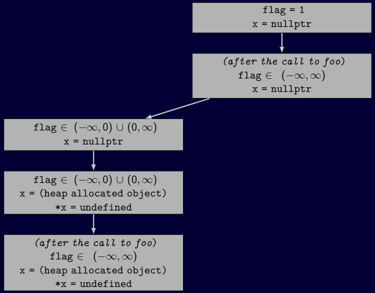
```
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
```

```
[B3]
x = new int;
```

```
[B2]
foo();
if (flag)
```

```
[B1]
*x = 5
```

[B0 (EXIT)]



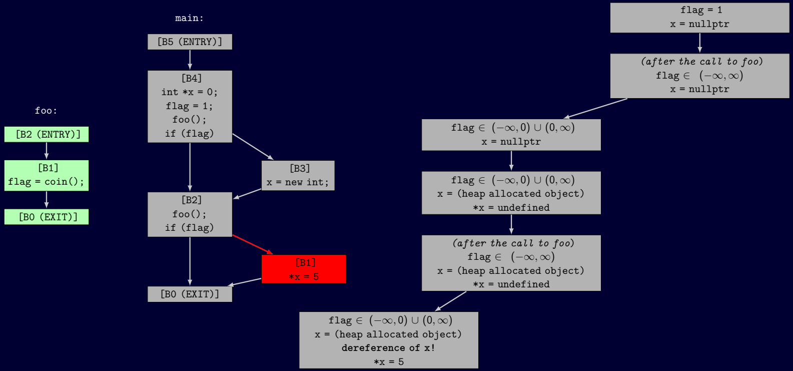
flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty,\infty)$
x = nullptr

flag ∈ $(-\infty,0) \cup (0,\infty)$
x = nullptr

flag ∈ $(-\infty,0) \cup (0,\infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty,\infty)$
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

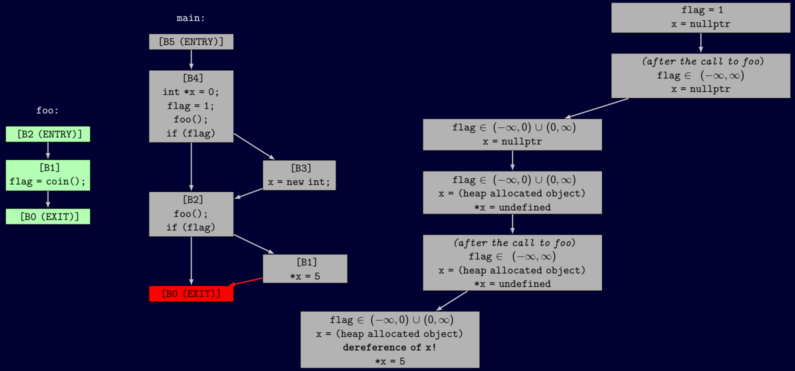
(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in (-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in (-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

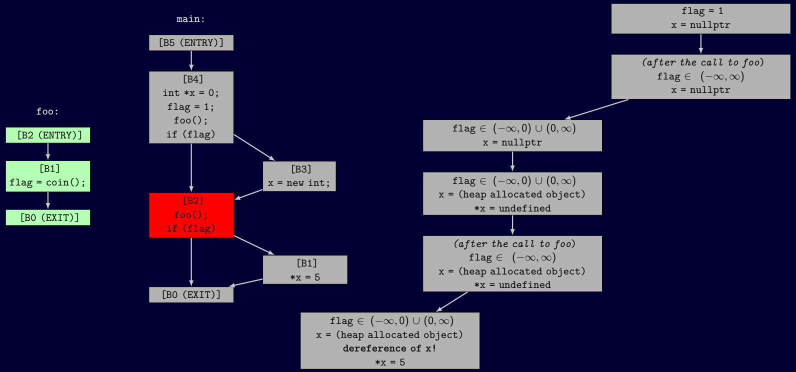
(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

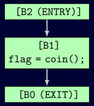
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

foo:

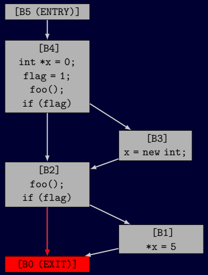


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

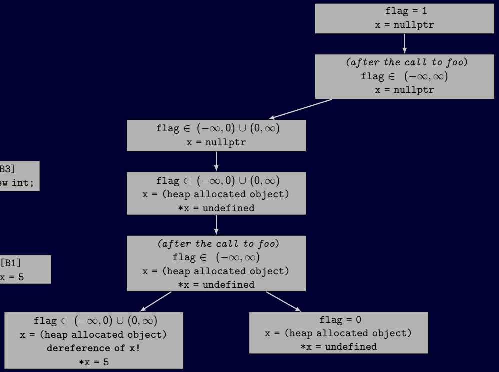
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

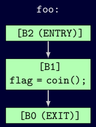
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

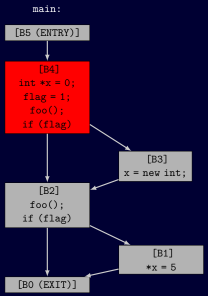


foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]



main:

[B5 (ENTRY)]

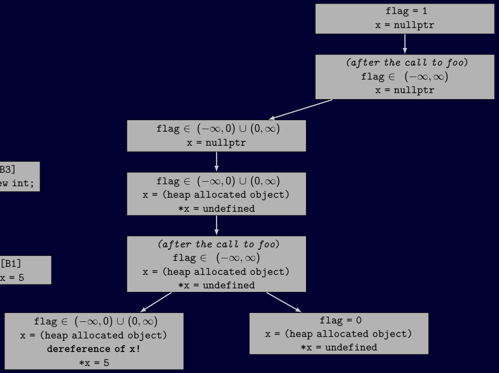
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

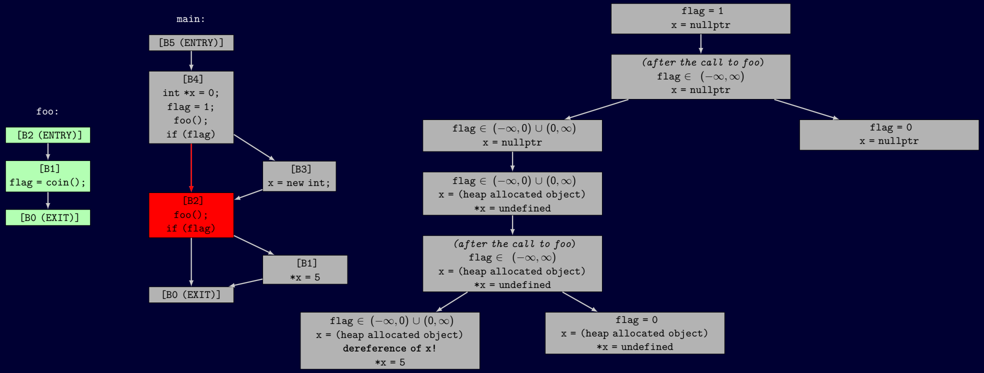
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

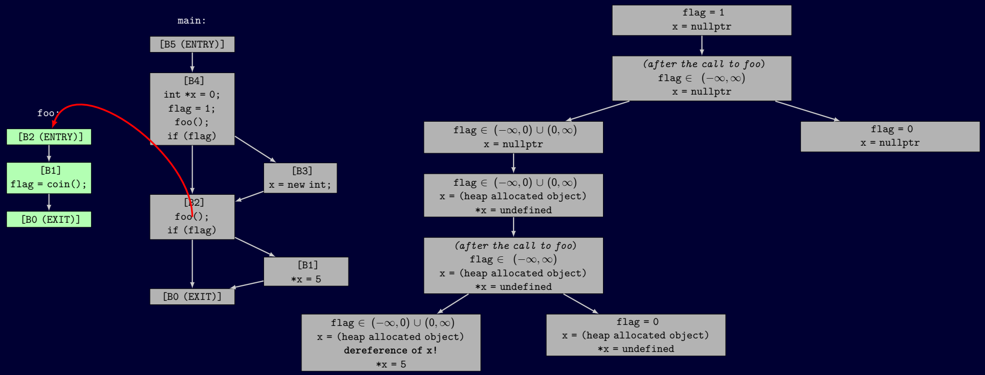
flag = 0
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

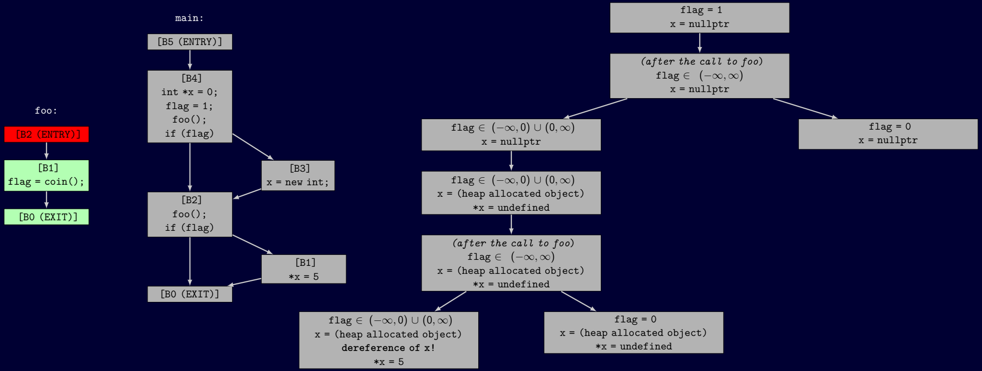
flag = 0
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag = 0
x = nullptr

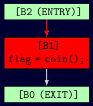
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

foo:

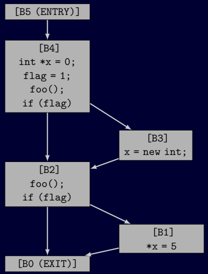


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:



[B5 (ENTRY)]

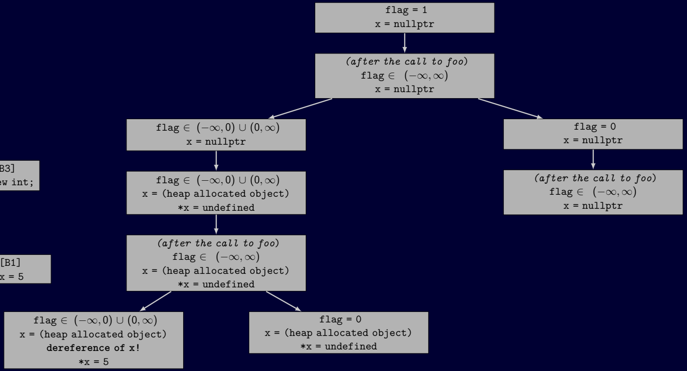
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

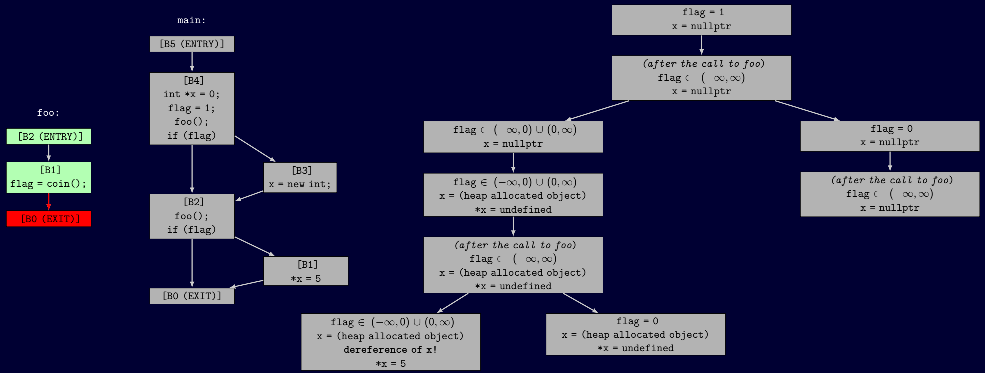
(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

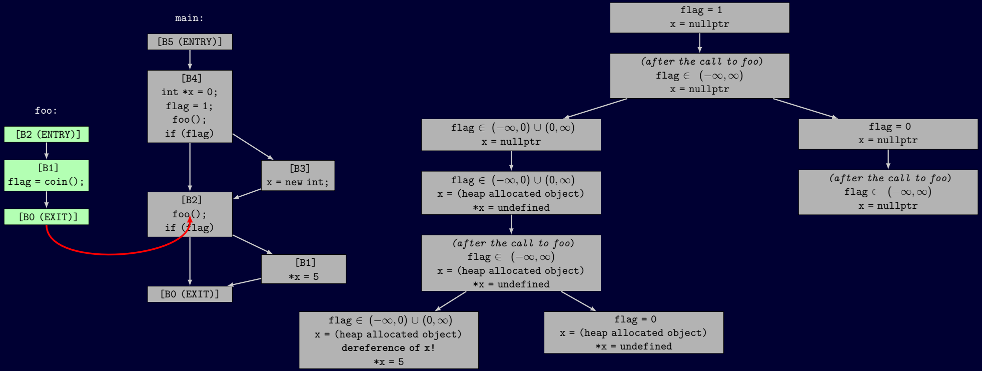
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

flag = 0
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr





foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag = 0
x = nullptr

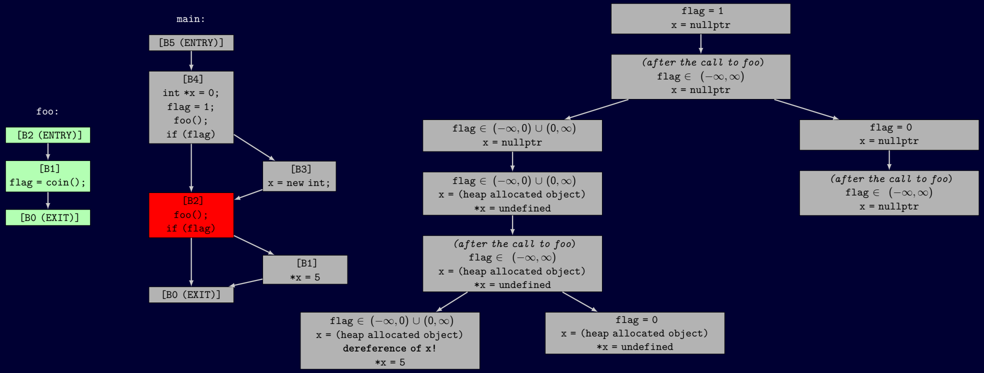
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

```
[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]
```

main:

```
[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]
```

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag = 0
x = nullptr

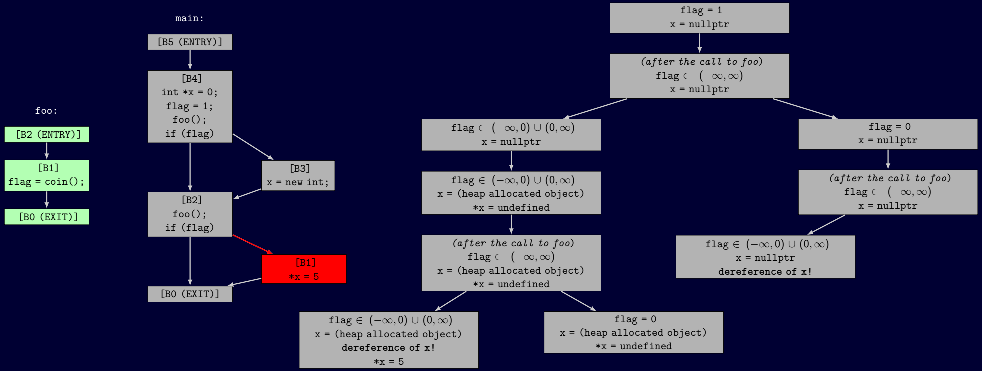
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag = 0
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

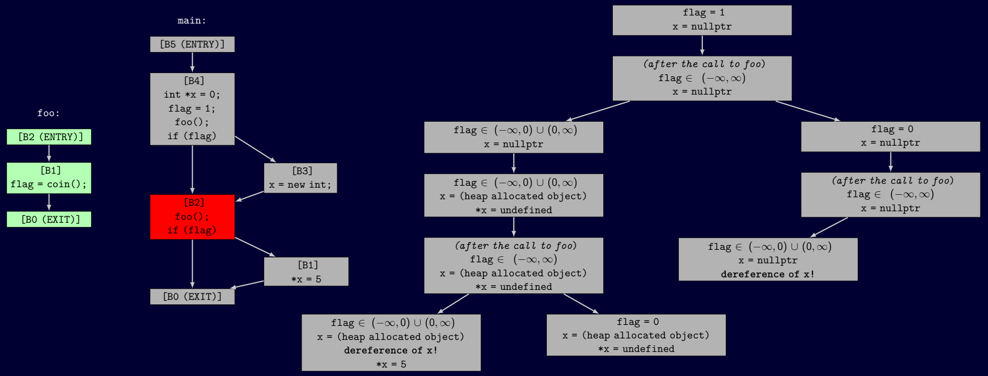
(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined



foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

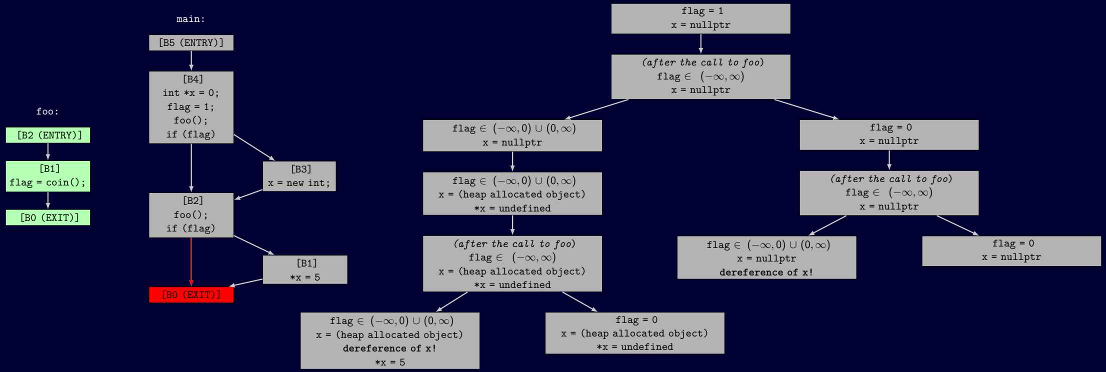
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

flag = 0
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



foo:

```
[B2 (ENTRY)]
[B1]
flag = coin();
[B0 (EXIT)]
```

main:

```
[B5 (ENTRY)]
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)
[B3]
x = new int;
[B2]
foo();
if (flag)
[B1]
*x = 5
[B0 (EXIT)]
```

flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

flag = 0
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 0
x = nullptr

There is always more to talk about...

- Branches in the ExplodedGraph may happen far more often

There is always more to talk about...

- Branches in the ExplodedGraph may happen far more often
- Representation of values, regions: symbols

There is always more to talk about...

- Branches in the ExplodedGraph may happen far more often
- Representation of values, regions: symbols
- ExplodedGraphs are usually very-very large, and contain tremendous amount of information

Bug report construction

Processing of the ExplodedGraph

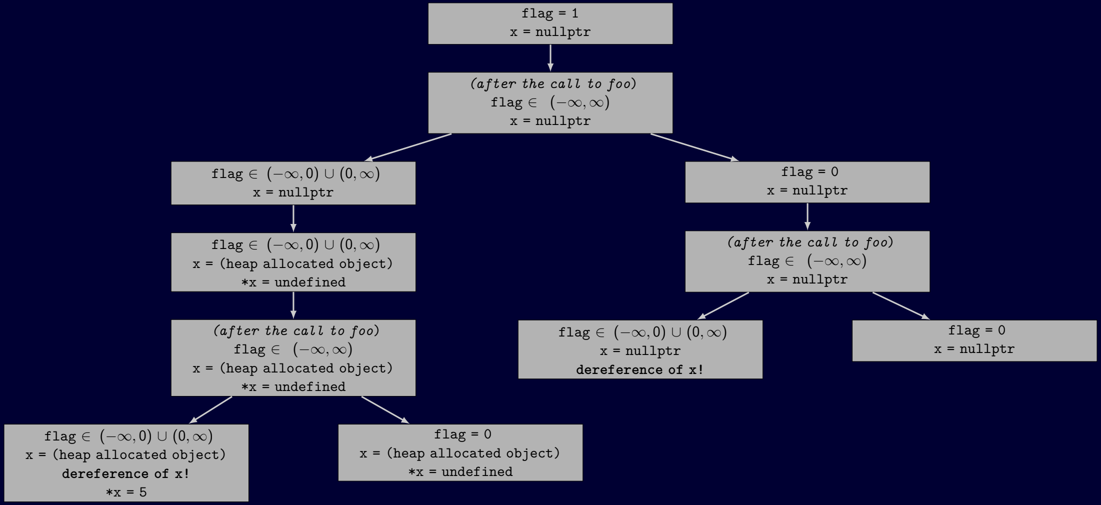
- Bugs are represented with error nodes
- The graph may contain several of them

Processing of the ExplodedGraph

- Bugs are represented with error nodes
- The graph may contain several of them
- The goal is to explain the path to these nodes

Processing of the ExplodedGraph

- Bugs are represented with error nodes
- The graph may contain several of them
- The goal is to explain the path to these nodes
- For each node, construct the shortest path from the root to the error node
- This is called a bug path



flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

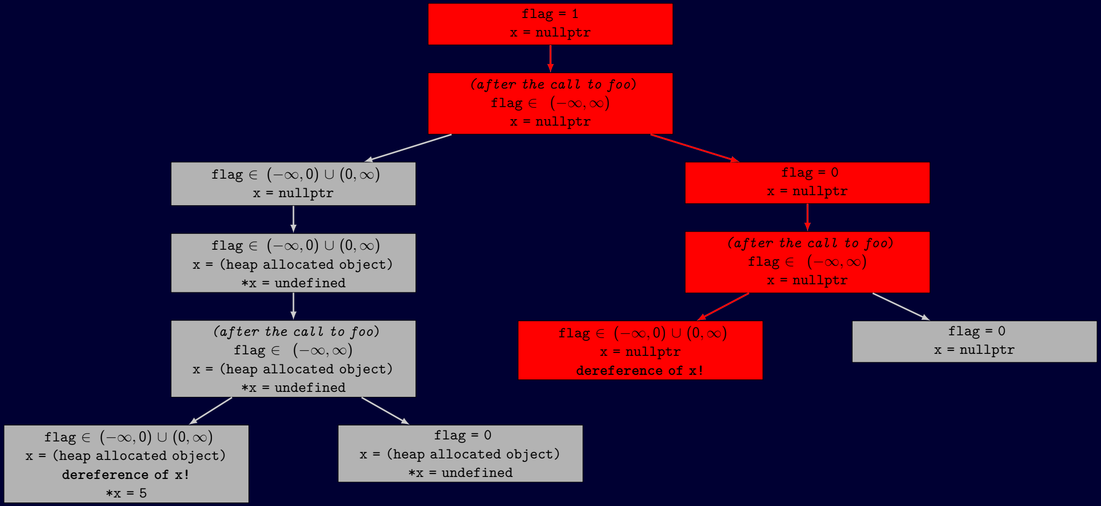
flag = 0
x = (heap allocated object)
*x = undefined

flag = 0
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 0
x = nullptr



flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
*x = undefined

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = (heap allocated object)
*x = undefined

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = (heap allocated object)
dereference of x!
*x = 5

flag = 0
x = (heap allocated object)
*x = undefined

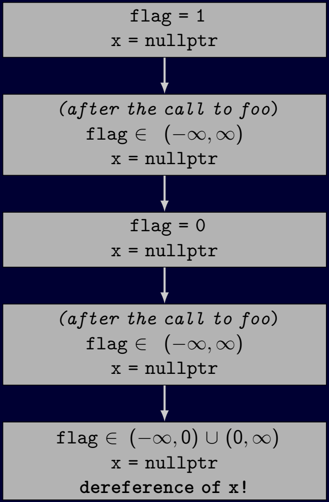
flag = 0
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 0
x = nullptr

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

The ideal bug report

The goal is to generate a bug report from the bug path that is

- complete: contains every information necessary to understand how the bug occured
- minimal: contains no unnecessary information

Techniques used by the analyzer

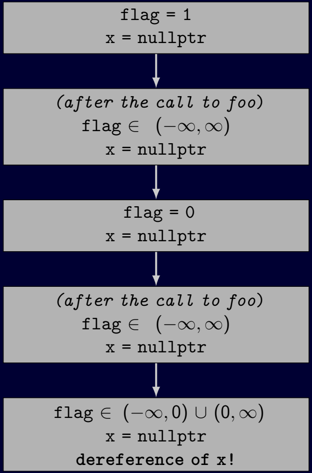
- 2 techniques:

Techniques used by the analyzer

- 2 techniques:
  - BugReporterVisitors
  - Interestingness propagation

Techniques used by the analyzer

- 2 techniques:
  - BugReporterVisitors
  - Interestingness propagation
- Visit the nodes of the bugpath from the error node to the root



flag = 1
x = nullptr

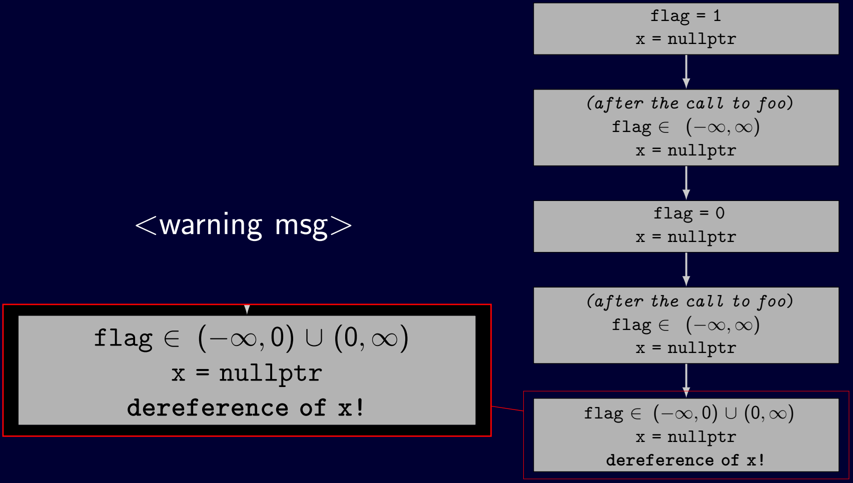
(after the call to foo)
flag $\in \ (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(after the call to foo)
flag $\in \ (-\infty, \infty)$
x = nullptr

flag $\in \ (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

<warning msg>



flag = 1
x = nullptr

(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

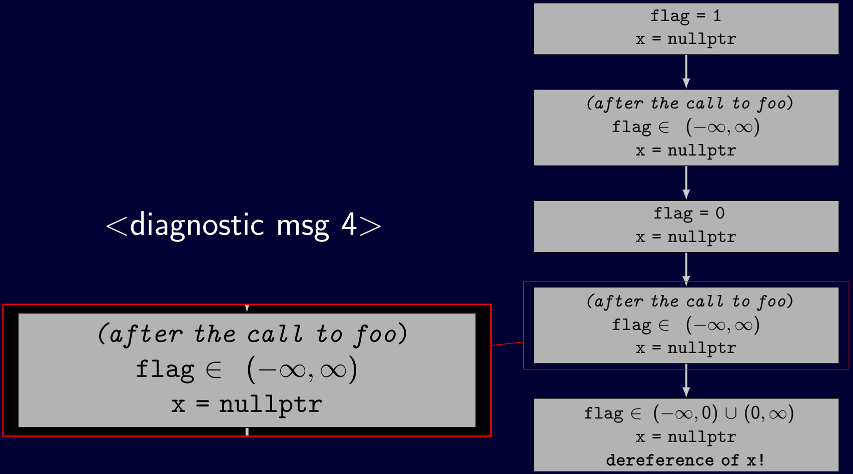
flag = 0
x = nullptr

(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

<diagnostic msg 4>



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

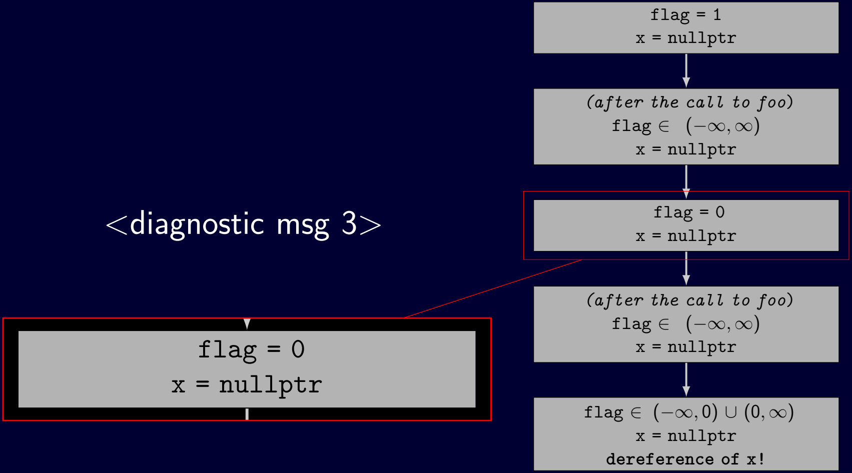
flag = 0
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

<diagnostic msg 3>



flag = 1
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

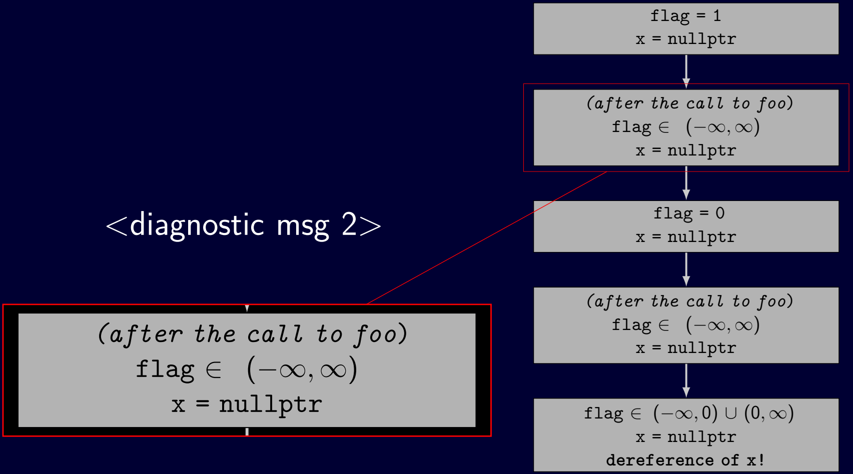
flag = 0
x = nullptr

(after the call to foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 0
x = nullptr

<diagnostic msg 2>



flag = 1
x = nullptr

(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

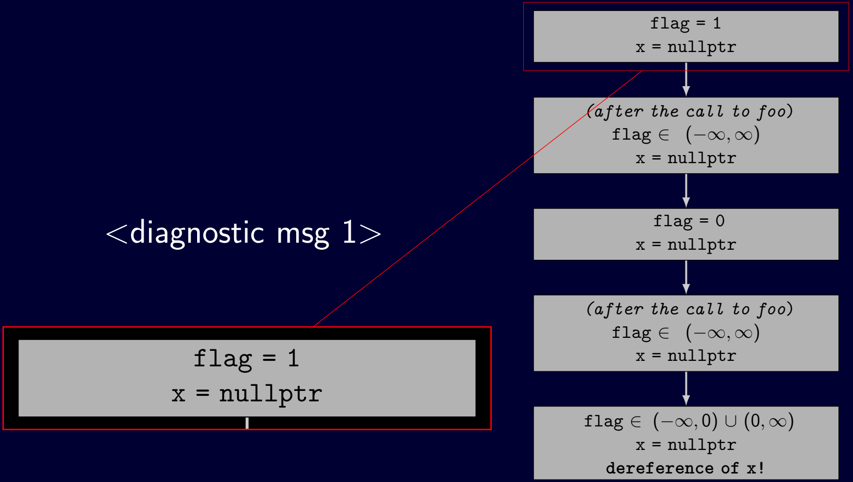
flag = 0
x = nullptr

(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(after the call to foo)
flag $\in (-\infty, \infty)$
x = nullptr

<diagnostic msg 1>



flag = 1
x = nullptr

(*after the call to foo*)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(*after the call to foo*)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 1
x = nullptr

BugReporterVisitors

- An arbitrary number of visitors can be registered

BugReporterVisitors

- An arbitrary number of visitors can be registered
- Their `visitNode()` is called on each node visit

BugReporterVisitors

- An arbitrary number of visitors can be registered
- Their `visitNode()` is called on each node visit
- Visitors may create diagnostic messages about the node they are currently visiting

BugReporterVisitors

- An arbitrary number of visitors can be registered
- Their `visitNode()` is called on each node visit
- Visitors may create diagnostic messages about the node they are currently visiting
- Despite the misleading name, they are more like callbacks than visitors

Visitors

- `ConditionBRVisitor`: Describes conditions of if branches, loops, conditional operators etc.

Visitors

- `ConditionBRVisitor`: Describes conditions of if branches, loops, conditional operators etc.
- `FindLastStoreBRVisitor`: Finds the last store to a given variable

Visitors

- `ConditionBRVisitor`: Describes conditions of if branches, loops, conditional operators etc.
- `FindLastStoreBRVisitor`: Finds the last store to a given variable
- `TrackControlDependencyCondBRVisitor`

TrackControlDependencyCondBRVisitor

- Most recent addition, available in Clang 10.0.0
- GSoC'19 project mentored by Artem Dergachev, Gábor Horváth and Ádám Balogh
- `https://szelethus.github.io/gsoc2019/`

TrackControlDependencyCondBRVisitor

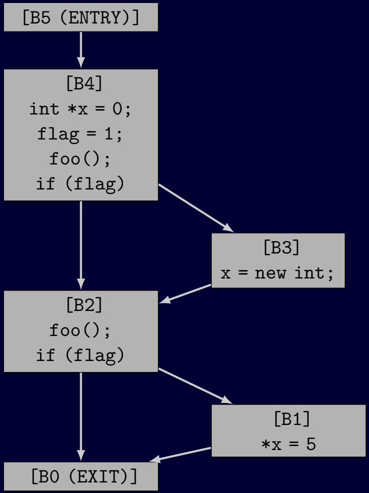
- Most recent addition, available in Clang 10.0.0
- GSoC'19 project mentored by Artem Dergachev, Gábor Horváth and Ádám Balogh
- `https://szelethus.github.io/gsoc2019/`
- Calculates control dependencies to points of interest

TrackControlDependencyCondBRVisitor

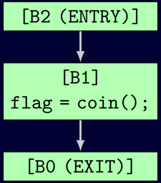
- Most recent addition, available in Clang 10.0.0
- GSoC'19 project mentored by Artem Dergachev, Gábor Horváth and Ádám Balogh
- `https://szelethus.github.io/gsoc2019/`
- Calculates control dependencies to points of interest
- Tells the analyzer to explain the conditions of control dependency blocks

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

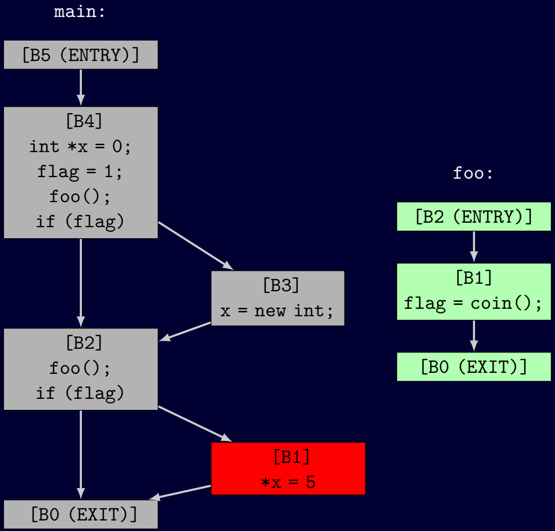
main:



foo:



```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

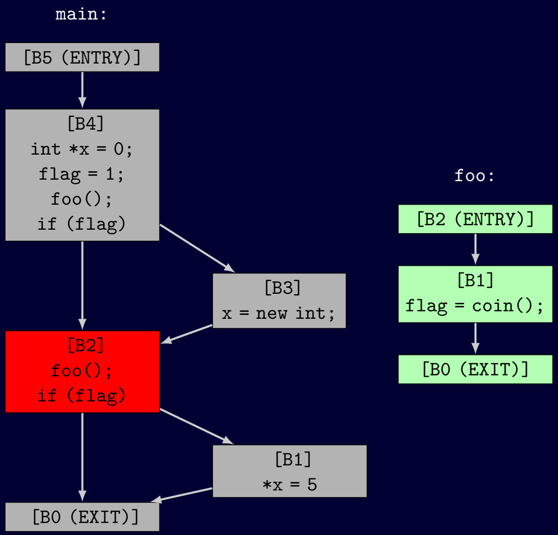
foo:

[B2 (ENTRY)]

[B1]
flag = coin();

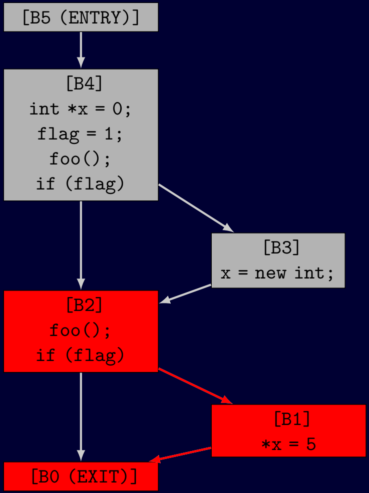
[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

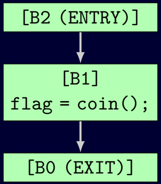


```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:

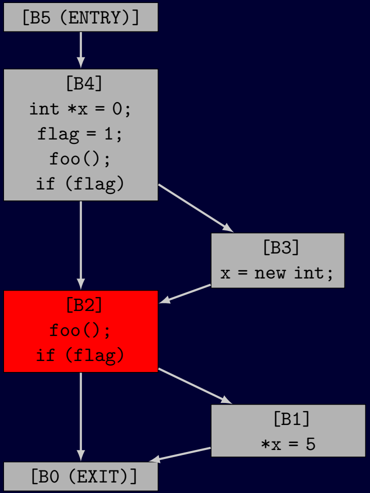


foo:



```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

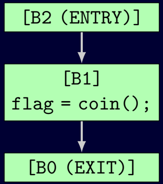
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:

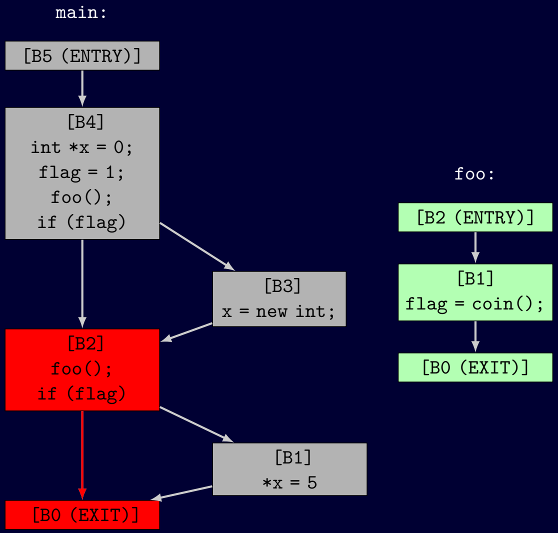


[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



Interestingness propagation

- During analysis, some symbolic regions or values may have been marked as ”interesting”.

Interestingness propagation

- During analysis, some symbolic regions or values may have been marked as ”interesting”.
- During bug report construction, propagate interestingness to entities that interact with an interesting entity

Interestingness propagation

- During analysis, some symbolic regions or values may have been marked as "interesting".
- During bug report construction, propagate interestingness to entities that interact with an interesting entity
- Nodes in the bug path that do not desribe an interesting entity are pruned

Combining Visitors and Interestingness

Expression value tracking!

Combining Visitors and Interestingness

Expression value tracking!

- Mark the expression as interesting

Combining Visitors and Interestingness

Expression value tracking!

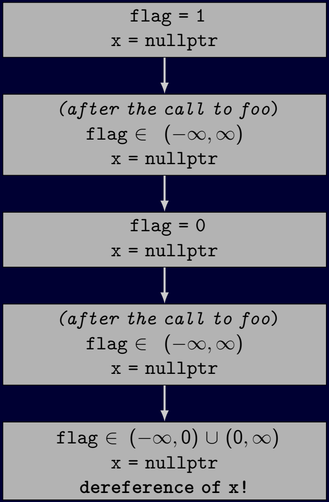
- Mark the expression as interesting
- Register visitors to describe events related to it
  - `FindLastStoreBRVisitor`
  - `TrackControlDependencyCondBRVisitor`
  - `ReturnVisitor`
  - `UndefOrNullArgVisitor`
  - etc...

Combining Visitors and Interestingness

Expression value tracking!

- Mark the expression as interesting
- Register visitors to describe events related to it
  - `FindLastStoreBRVisitor`
  - `TrackControlDependencyCondBRVisitor`
  - `ReturnVisitor`
  - `UndefOrNullArgVisitor`
  - etc...
- `TrackControlDependencyCondBRVisitor` does that as well

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



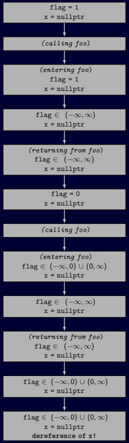
flag = 1
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(after the call to foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

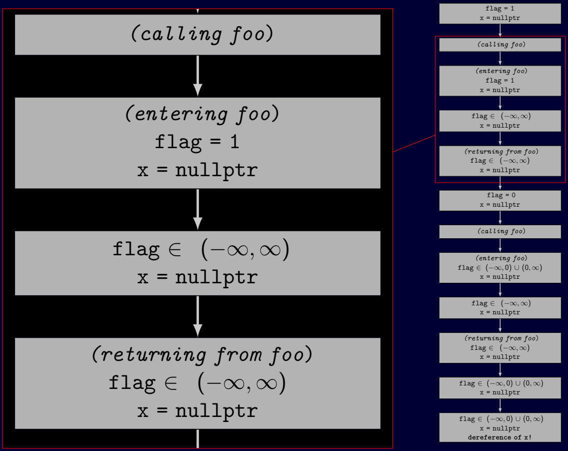
(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

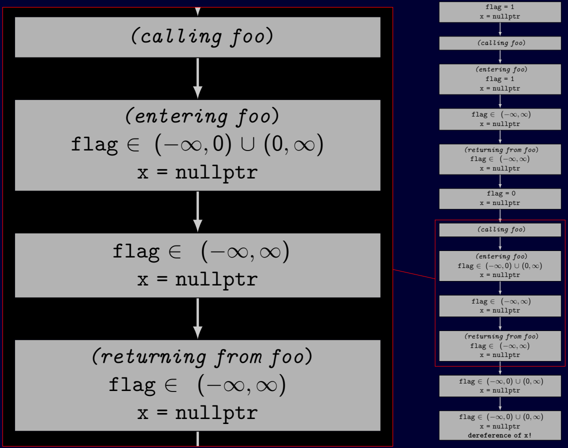
(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

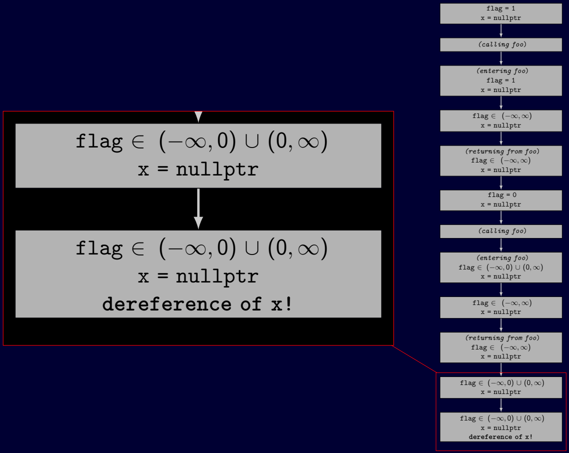
(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

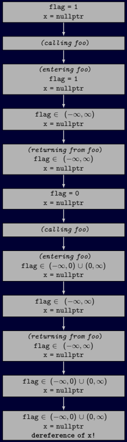
flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

Stage 1: Visitor notes



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

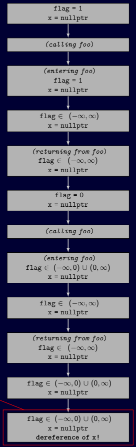
The checker tracks x's value

$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$

x = nullptr

dereference of x!

Tracked variables: $\{\texttt{x}\}$



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

(returning from foo)
$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr

$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

(returning from foo)
$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

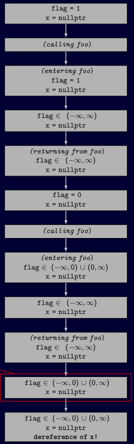
$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr

$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

ConditionBRVisitor: Assuming 'flag' is not equal to 0

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

Tracked variables: $\{x\}$



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

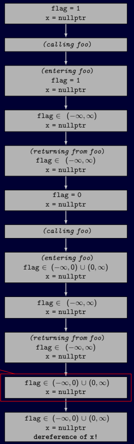
(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: Assuming flag is true

`TrackControlDependencyCondBRVisitor` tracks `flag`


flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

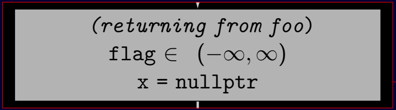
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: Assuming `flag` is true

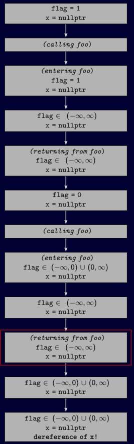
`flag` $\in$ $(-\infty, 0) \cup (0, \infty)$
`x = nullptr`

Tracked variables: {`x`, `flag`}



(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

Tracked variables: {x, flag}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

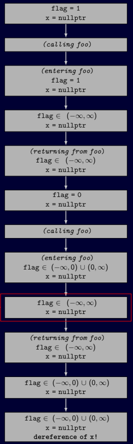
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: Assuming flag is true

FindLastStoreBRVisitor:
Value assigned to 'flag'

flag $\in (-\infty, \infty)$
x = nullptr

Tracked variables: {x, flag}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: flag is assigned a value

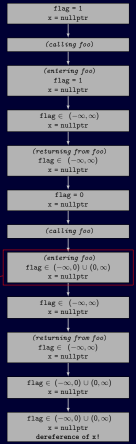
note: Assuming flag is true

(entering foo)

flag $\in$ $(-\infty, 0) \cup (0, \infty)$

x = nullptr

Tracked variables: {x, flag}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

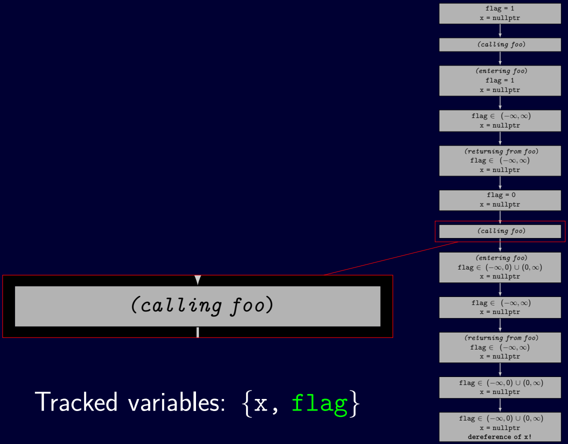
(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: flag is assigned a value

note: Assuming flag is true



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

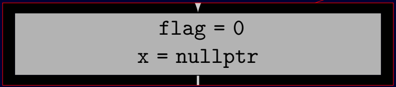
(calling foo)

note: `flag` is assigned a value

note: Assuming `flag` is true

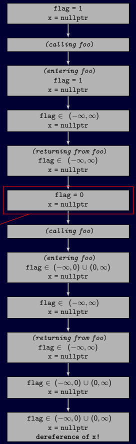
Tracked variables: {`x`, `flag`}

ConditionBRVisitor: Assuming 'flag' is 0



flag = 0
x = nullptr

Tracked variables: {x, flag}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

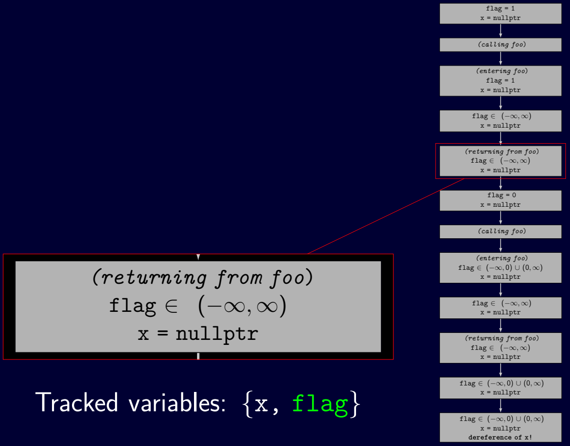
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: Assuming flag is false

note: flag is assigned a value

note: Assuming flag is true



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

note: Assuming `flag` is false

note: `flag` is assigned a value

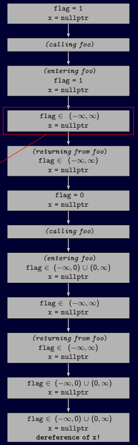
note: Assuming `flag` is true

Tracked variables: {`x`, `flag`}

`FindLastStoreBRVisitor` is already satisfied, no notes

$\texttt{flag} \in (-\infty, \infty)$
`x = nullptr`

Tracked variables: {`x`, `flag`}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

(returning from foo)
$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

note: Assuming `flag` is false

(calling foo)

(entering foo)
$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr

$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

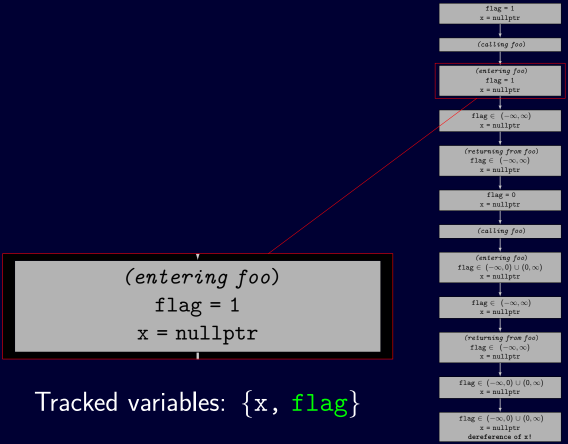
note: `flag` is assigned a value

(returning from foo)
$\texttt{flag} \in (-\infty, \infty)$
x = nullptr

$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr

note: Assuming `flag` is true

$\texttt{flag} \in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

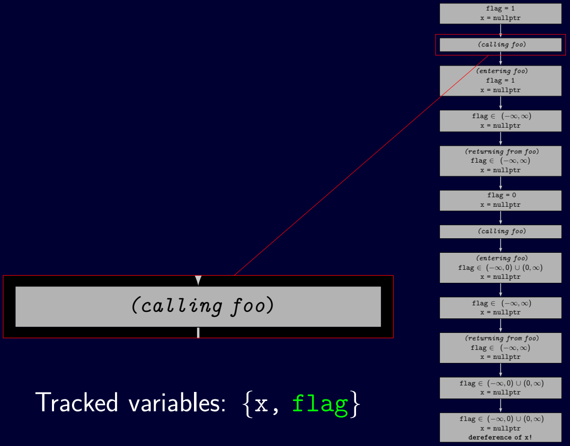
(entering foo)
flag = 1
x = nullptr

note: Assuming `flag` is false

note: `flag` is assigned a value

note: Assuming `flag` is true

Tracked variables: {`x`, `flag`}



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(calling foo)

note: Assuming `flag` is false

note: `flag` is assigned a value

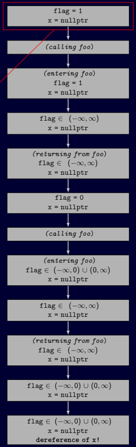
note: Assuming `flag` is true

Tracked variables: {`x`, `flag`}

`FindLastStoreBRVisitor`:
'x' initialized to null pointer value

```
flag = 1
x = nullptr
```

Tracked variables: {`x`, `flag`}



```
flag = 1
x = nullptr
```

note: `x` initialized to nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

note: Assuming `flag` is false

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

note: `flag` is assigned a value

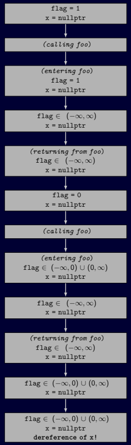
(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

note: Assuming `flag` is true

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

Stage 2: Non-visitor notes



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

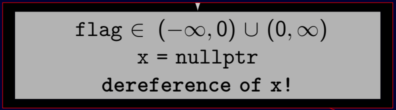
note: `x` initialized to nullptr

note: Assuming `flag` is false

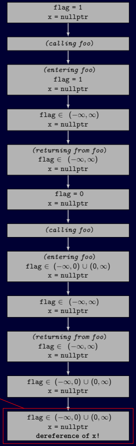
note: `flag` is assigned a value

note: Assuming `flag` is true

The warning message is supplied by the checker



flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

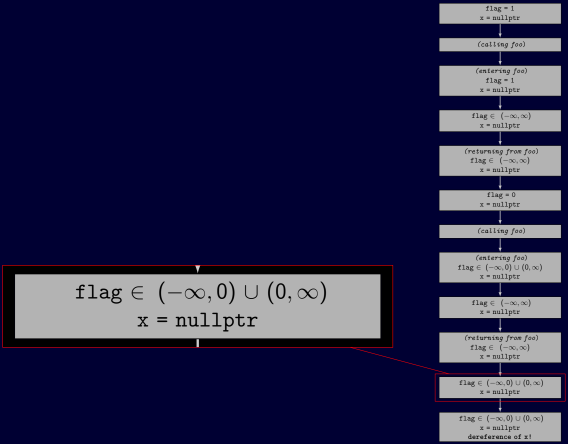
note: x initialized to nullptr

note: Assuming flag is false

note: flag is assigned a value

note: Assuming flag is true

warning: Nullptr dereference



flag = 1
x = nullptr

note: x initialized to nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

note: Assuming `flag` is false

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

note: `flag` is assigned a value

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

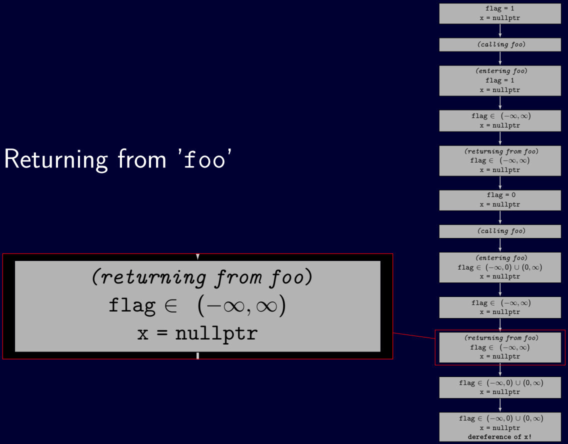
note: Assuming `flag` is true

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

warning: Nullptr dereference

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

Returning from 'foo'



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

note: x initialized to nullptr

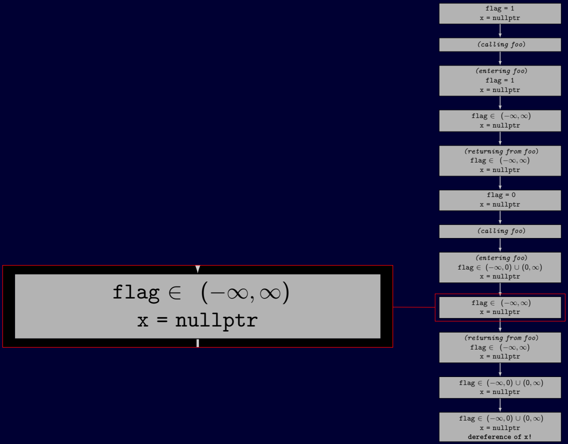
note: Assuming flag is false

note: flag is assigned a value

note: Returning from foo

note: Assuming flag is true

warning: Nullptr dereference



flag = 1
x = nullptr

note: x initialized to nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

note: Assuming `flag` is false

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

note: `flag` is assigned a value

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

note: Returning from `foo`

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

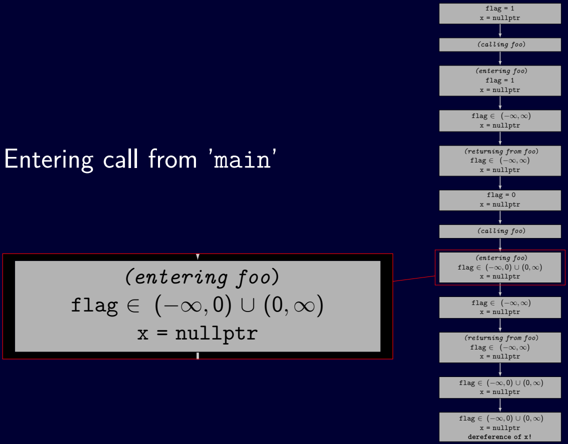
note: Assuming `flag` is true

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

warning: Nullptr dereference

flag ∈ $(-\infty, \infty)$
x = nullptr

Entering call from 'main'



(*entering foo*)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag = 1
x = nullptr

(*calling foo*)

(*entering foo*)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(*returning from foo*)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(*calling foo*)

(*entering foo*)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(*returning from foo*)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: x initialized to nullptr

note: Assuming flag is false

note: Entered function from main

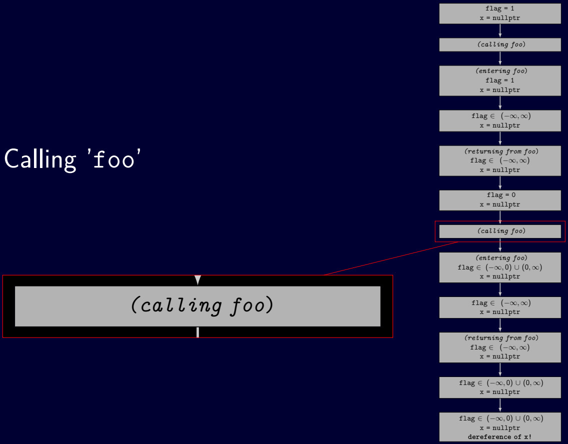
note: flag is assigned a value

note: Returning from foo

note: Assuming flag is true

warning: Nullptr dereference

Calling 'foo'



(calling foo)

flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: x initialized to nullptr

note: Assuming flag is false

note: Calling foo

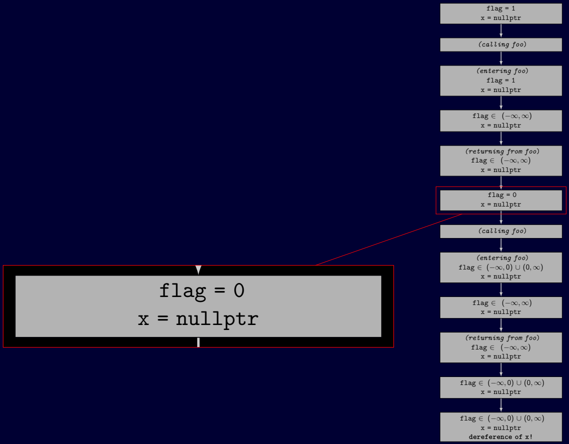
note: Entered function from main

note: flag is assigned a value

note: Returning from foo

note: Assuming flag is true

warning: Nullptr dereference



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag = 0
x = nullptr

note: `x` initialized to nullptr

note: Assuming `flag` is false

note: Calling `foo`

note: Entered function from `main`

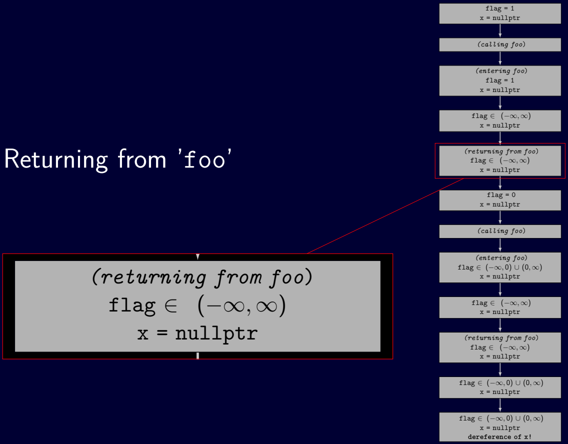
note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference

Returning from 'foo'



(*returning from foo*)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 1
x = nullptr

(*calling foo*)

(*entering foo*)
flag = 1
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(*returning from foo*)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(*calling foo*)

(*entering foo*)
flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, \infty)$
x = nullptr

(*returning from foo*)
flag $\in$ $(-\infty, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in$ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: `x` initialized to nullptr

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

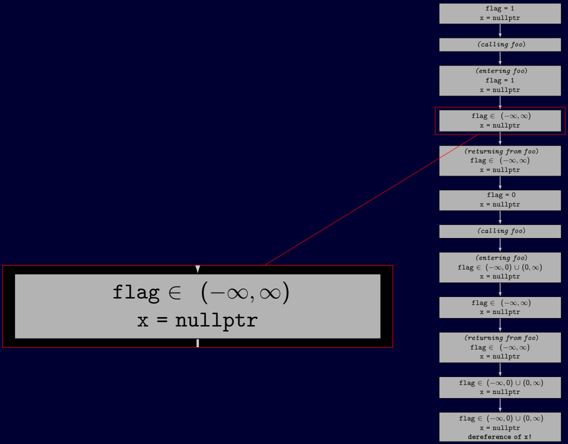
note: Entered function from `main`

note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

flag $\in (-\infty, \infty)$
x = nullptr

note: x initialized to nullptr

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

note: Entered function from `main`

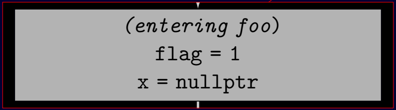
note: `flag` is assigned a value

note: Returning from `foo`

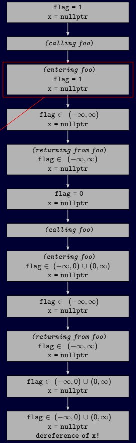
note: Assuming `flag` is true

warning: Nullptr dereference

Entering call from 'main'



(entering foo)
flag = 1
x = nullptr



flag = 1
x = nullptr

note: x initialized to nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

note: Entered function from main

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

note: Returning from foo

flag = 0
x = nullptr

note: Assuming flag is false

(calling foo)

note: Calling foo

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

note: Entered function from main

flag $\in (-\infty, \infty)$
x = nullptr

note: flag is assigned a value

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

note: Returning from foo

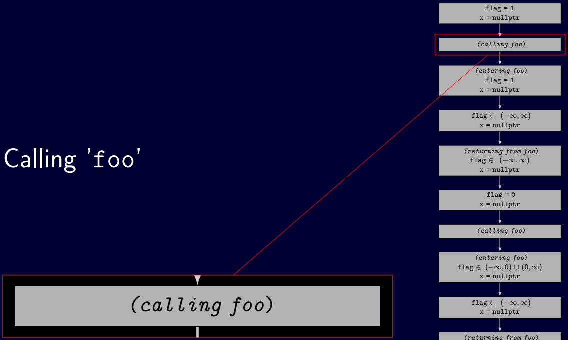
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

note: Assuming flag is true

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

warning: Nullptr dereference

Calling 'foo'



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

(calling foo)

note: x initialized to nullptr

note: Calling `foo`

note: Entered function from `main`

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

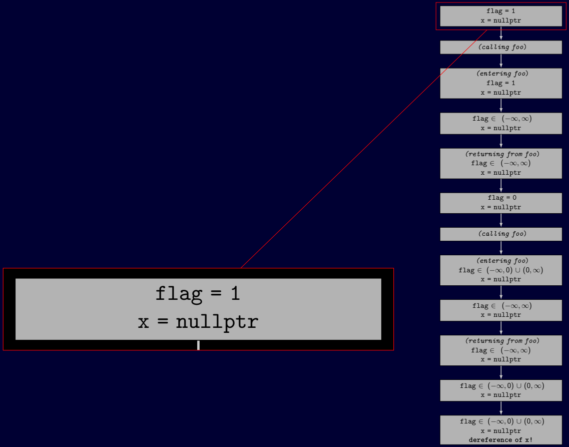
note: Entered function from `main`

note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference



flag = 1
x = nullptr

flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: `x` initialized to nullptr

note: Calling `foo`

note: Entered function from `main`

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

note: Entered function from `main`

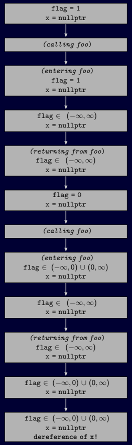
note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference

Stage 3: Pruning



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, \infty)$
x = nullptr

(returning from foo)
flag ∈ $(-\infty, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr

flag ∈ $(-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: `x` initialized to nullptr

note: Calling `foo`

note: Entered function from `main`

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

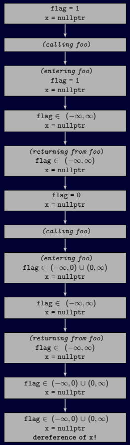
note: Entered function from `main`

note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: `x` initialized to nullptr

note: Calling `foo`

note: Entered function from `main`

note: Returning from `foo`

note: Assuming `flag` is false

note: Calling `foo`

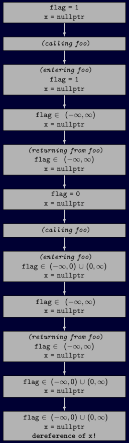
note: Entered function from `main`

note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference



flag = 1
x = nullptr

(calling foo)

(entering foo)
flag = 1
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag = 0
x = nullptr

(calling foo)

(entering foo)
flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, \infty)$
x = nullptr

(returning from foo)
flag $\in (-\infty, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr

flag $\in (-\infty, 0) \cup (0, \infty)$
x = nullptr
dereference of x!

note: `x` initialized to nullptr

note: Assuming `flag` is false

note: Calling `foo`

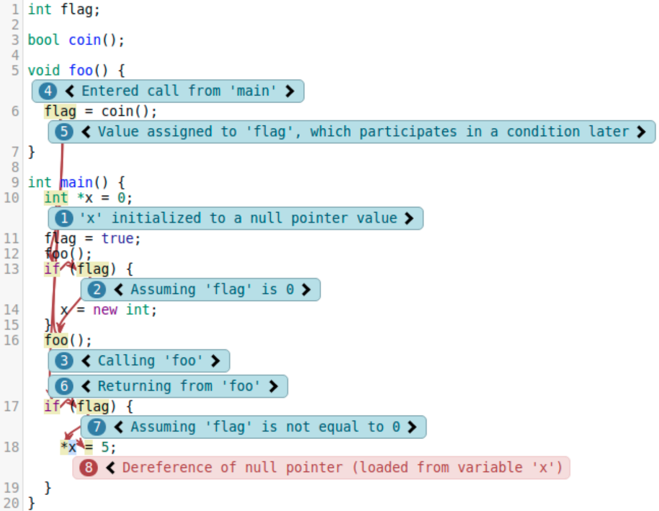
note: Entered function from `main`

note: `flag` is assigned a value

note: Returning from `foo`

note: Assuming `flag` is true

warning: Nullptr dereference



```
int flag;

bool coin();

void foo() {
  4 < Entered call from 'main' >
  flag = coin();
    5 < Value assigned to 'flag', which participates in a condition later >
}

int main() {
  int *x = 0;
    1 'x' initialized to a null pointer value >
  flag = true;
  foo();
  if (flag) {
      2 < Assuming 'flag' is 0 >
    x = new int;
  }
  foo();
    3 < Calling 'foo' >
    6 < Returning from 'foo' >
  if (flag) {
      7 < Assuming 'flag' is not equal to 0 >
    *x = 5;
      8 < Dereference of null pointer (loaded from variable 'x')
  }
}
```

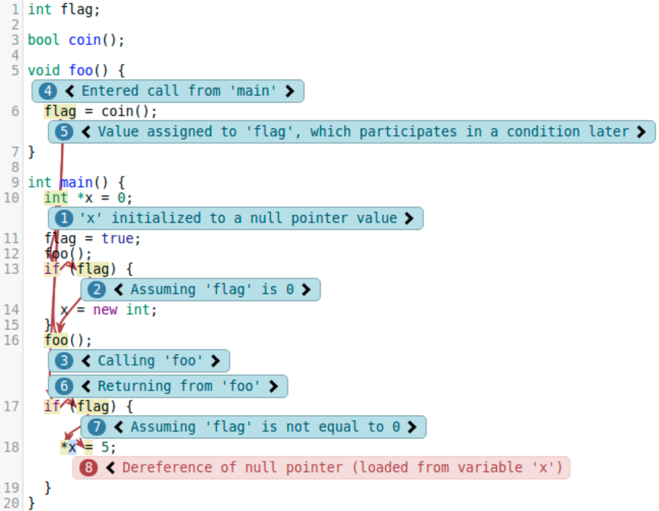
Present problems, research

Very hard to solve problems

- Relevant information isn’t found in the bug path

Very hard to solve problems

- Relevant information isn't found in the bug path
- Ad absurdum, not even in the ExplodedGraph



```
int flag;

bool coin();

void foo() {
  4 < Entered call from 'main' >
  flag = coin();
  5 < Value assigned to 'flag', which participates in a condition later >
}

int main() {
  int *x = 0;
  1 'x' initialized to a null pointer value >
  flag = true;
  foo();
  if (flag) {
    2 < Assuming 'flag' is 0 >
    x = new int;
  }
  foo();
  3 < Calling 'foo' >
  6 < Returning from 'foo' >
  if (flag) {
    7 < Assuming 'flag' is not equal to 0 >
    *x = 5;
    8 < Dereference of null pointer (loaded from variable 'x')
  }
}
```

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

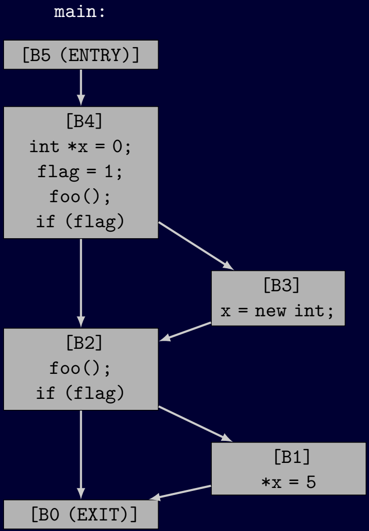
```
int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

```
int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    printf("Nothing to see here!");
  foo();

  if (flag)
    *x = 5;
}
```

Reaching definitions analysis

- An algorithm to find a set of last stores (definitions) to a variable
- Regard all definitions to a variable as a point of interest
- `https://reviews.llvm.org/D64991`

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

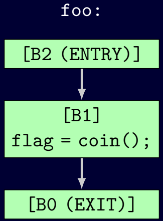
[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]



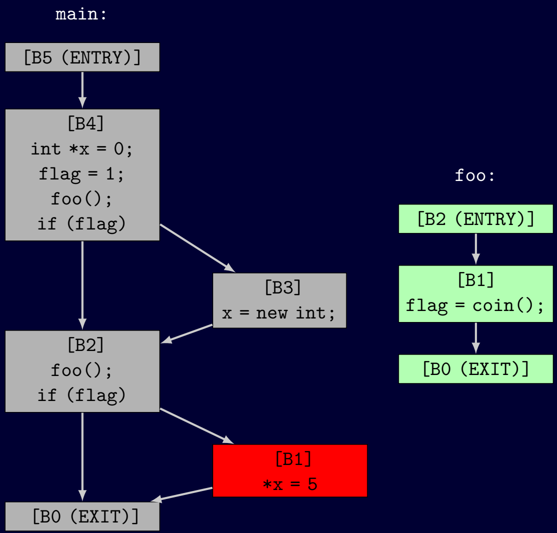
foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

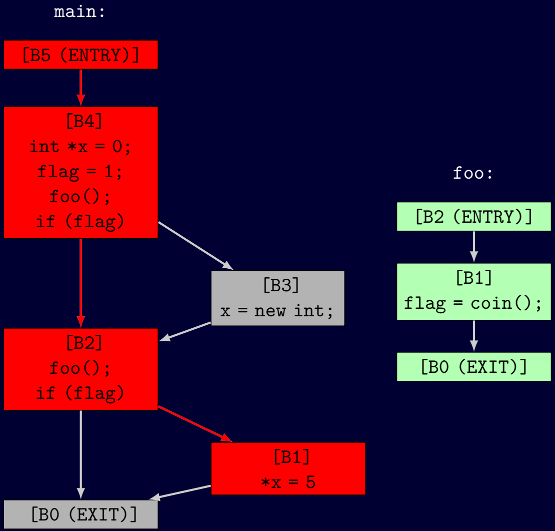
foo:

[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



main:

[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

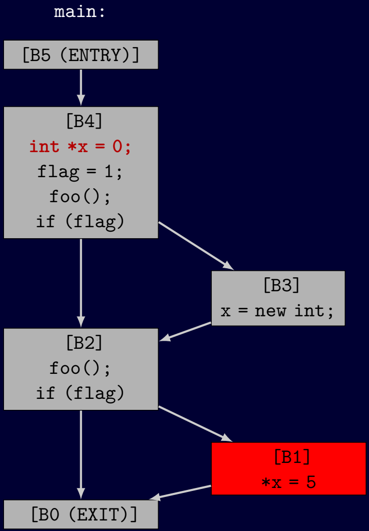
foo:

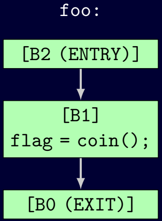
[B2 (ENTRY)]

[B1]
flag = coin();

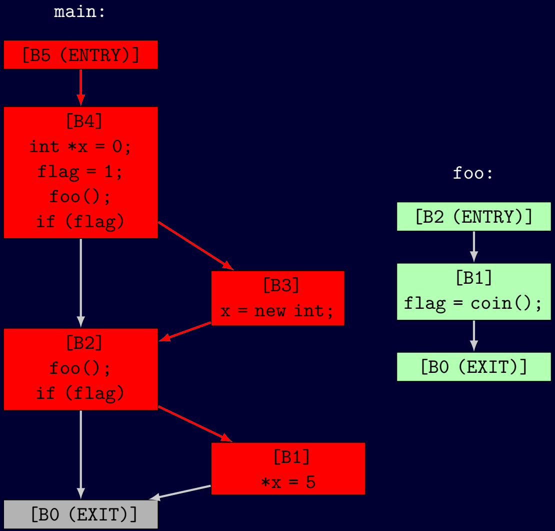
[B0 (EXIT)]

```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



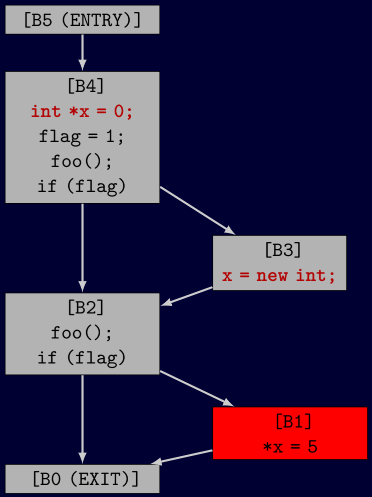


```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```



```
int flag;
bool coin();

void foo() {
  flag = coin();
}

int main() {
  int *x = 0;
  flag = 1;
  foo();
  if (flag)
    x = new int;
  foo();

  if (flag)
    *x = 5;
}
```

main:



[B5 (ENTRY)]

[B4]
int *x = 0;
flag = 1;
foo();
if (flag)

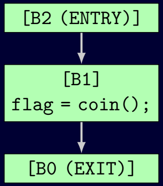
[B3]
x = new int;

[B2]
foo();
if (flag)

[B1]
*x = 5

[B0 (EXIT)]

foo:



[B2 (ENTRY)]

[B1]
flag = coin();

[B0 (EXIT)]

Problems with reaching definitions

- Originally concieved for instructions
- Incredibly complex to implement for C, C++, etc...
- Doesn't argue about aliasing
- Only works in a given CFG...

Problems with reaching definitions

- Originally concieved for instructions
- Incredibly complex to implement for C, C++, etc...
- Doesn't argue about aliasing
- Only works in a given CFG...
- Using visitors, its possible to make this algorithm semi-interprocedural

Conclusion

Conclusion

- Clear and precise bug reports are important

Conclusion

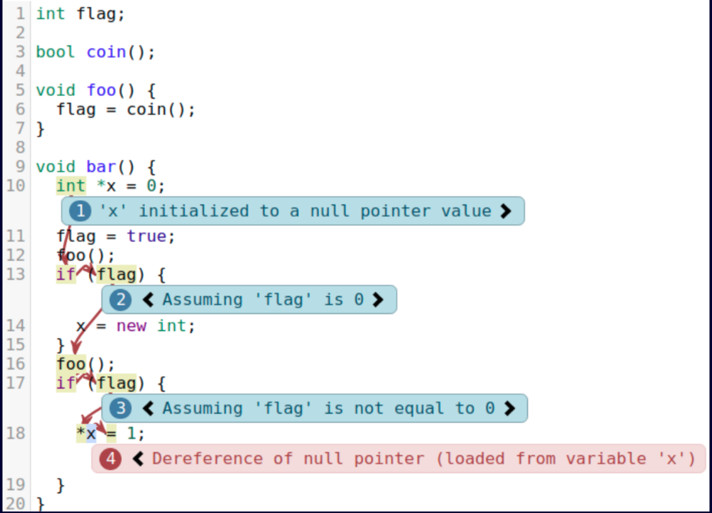
- Clear and precise bug reports are important
- The analyzer users callbacks (or visitors) and interestingness propagation to construct path-sensitive bug reports

Conclusion

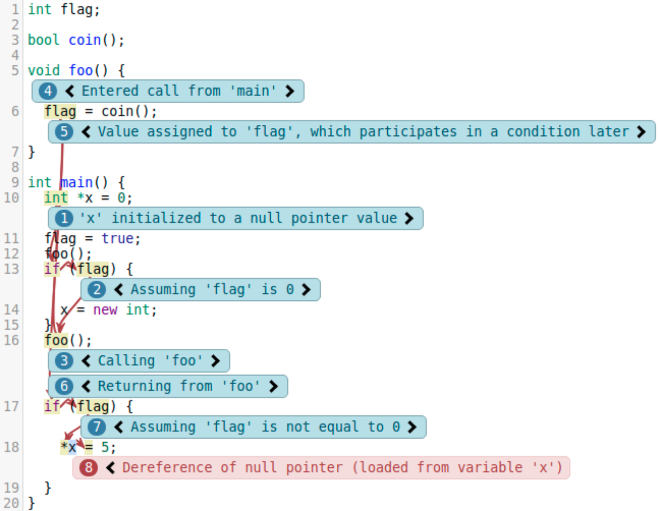
- Clear and precise bug reports are important
- The analyzer users callbacks (or visitors) and interestingness propagation to construct path-sensitive bug reports
- Problems that require arguing outside the bugpath, especially the ExplodedGraph are insanely difficult

Conclusion

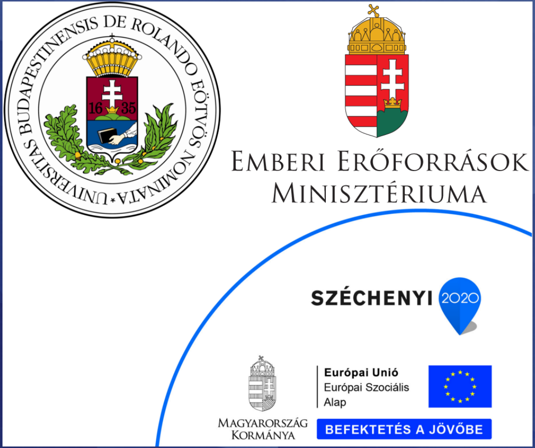
- Clear and precise bug reports are important
- The analyzer users callbacks (or visitors) and interestingness propagation to construct path-sensitive bug reports
- Problems that require arguing outside the bugpath, especially the ExplodedGraph are insanely difficult
- The analyzer gets better by the minute



```
int flag;

bool coin();

void foo() {
  flag = coin();
}

void bar() {
  int *x = 0;
   1 'x' initialized to a null pointer value ❯
  flag = true;
  foo();
  if (flag) {
       2 ❮ Assuming 'flag' is 0 ❯
    x = new int;
  }
  foo();
  if (flag) {
       3 ❮ Assuming 'flag' is not equal to 0 ❯
    *x = 1;
      4 ❮ Dereference of null pointer (loaded from variable 'x')
  }
}
```



```
int flag;

bool coin();

void foo() {
 4 < Entered call from 'main' >
  flag = coin();
   5 < Value assigned to 'flag', which participates in a condition later >
}

int main() {
  int *x = 0;
   1 'x' initialized to a null pointer value >
  flag = true;
  foo();
  if (flag) {
     2 < Assuming 'flag' is 0 >
    x = new int;
  }
  foo();
   3 < Calling 'foo' >
   6 < Returning from 'foo' >
  if (flag) {
     7 < Assuming 'flag' is not equal to 0 >
    *x = 5;
      8 < Dereference of null pointer (loaded from variable 'x')
  }
}
```



UNIVERSITAS BUDAPESTINENSIS DE ROLANDO EÖTVÖS NOMINATA

16 35

EMBERI ERŐFORRÁSOK
MINISZTÉRIUMA

SZÉCHENYI 2020

MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Szociális
Alap

BEFEKTETÉS A JÖVŐBE

Questions?