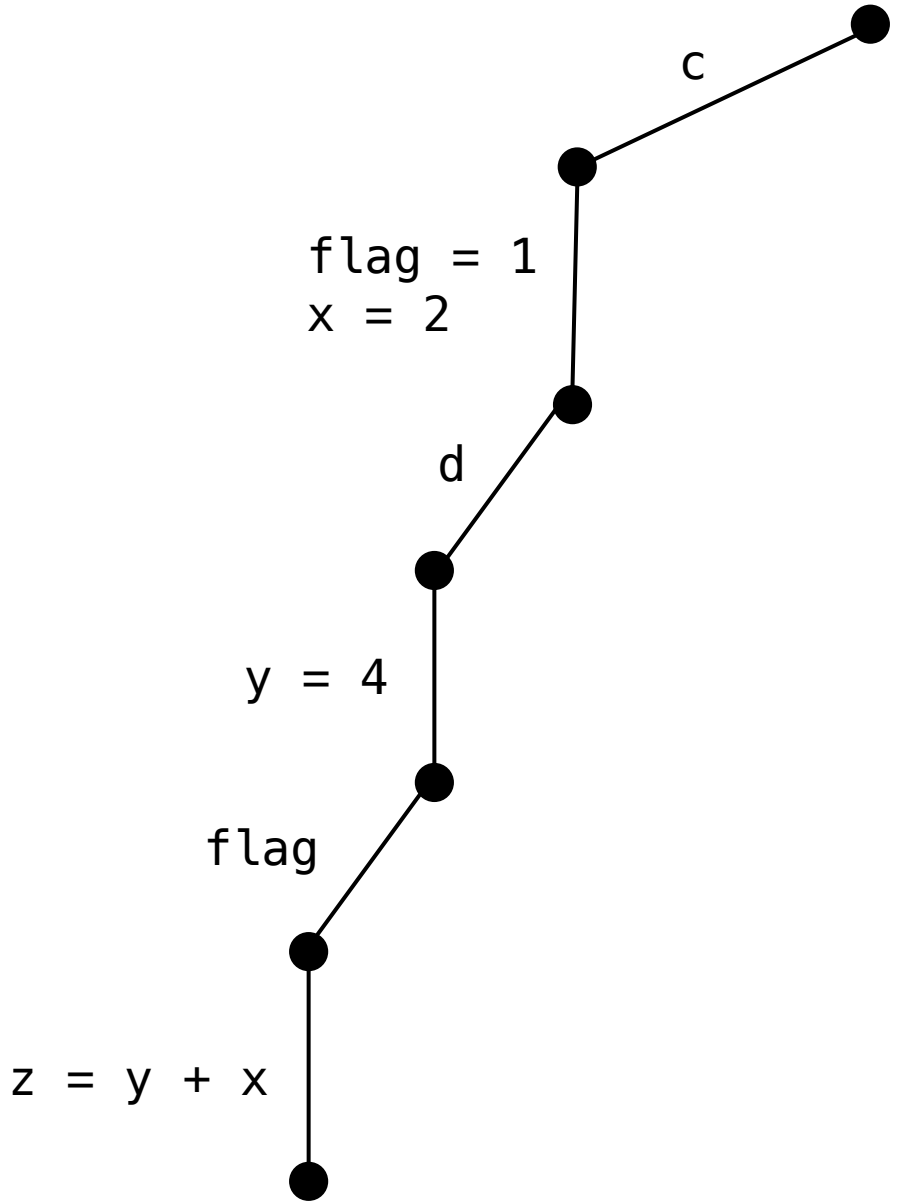
Example

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

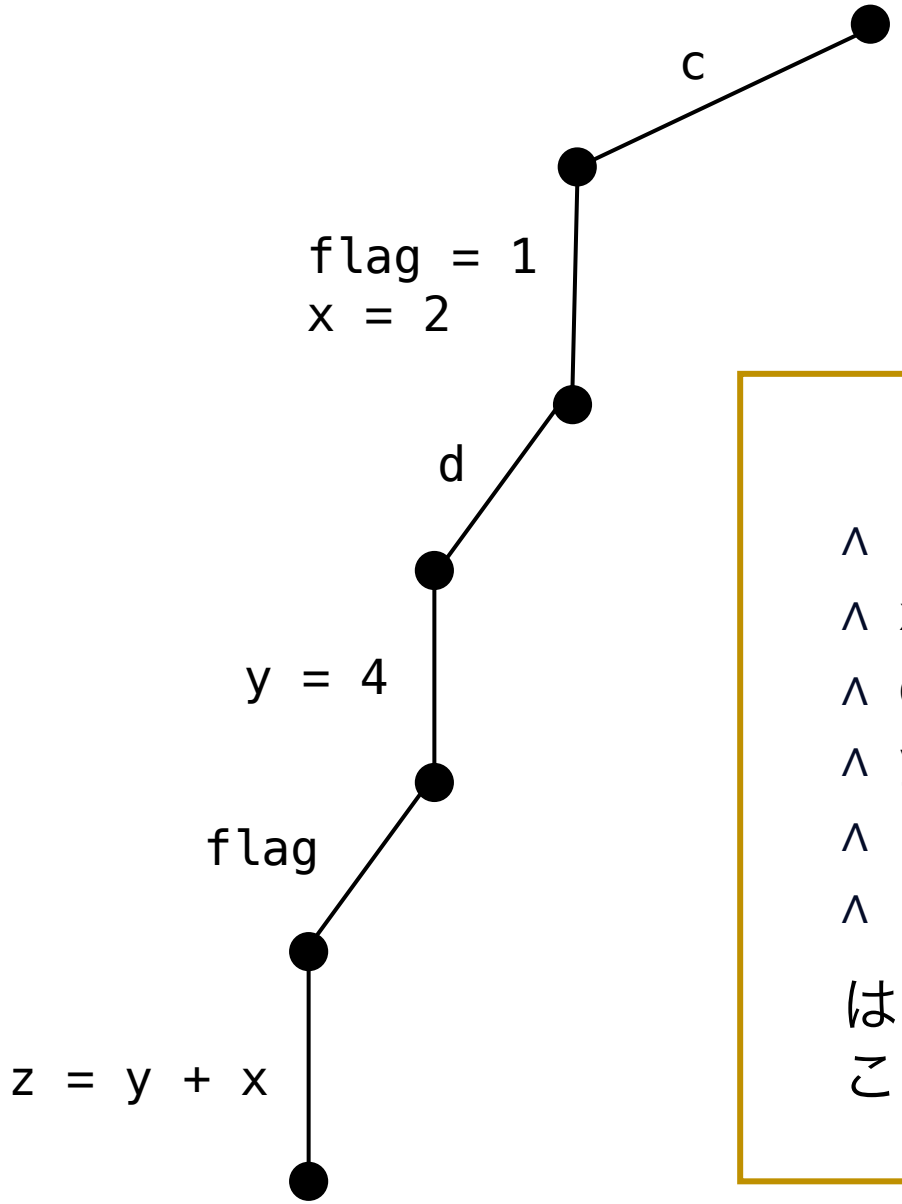
Example

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



Example

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

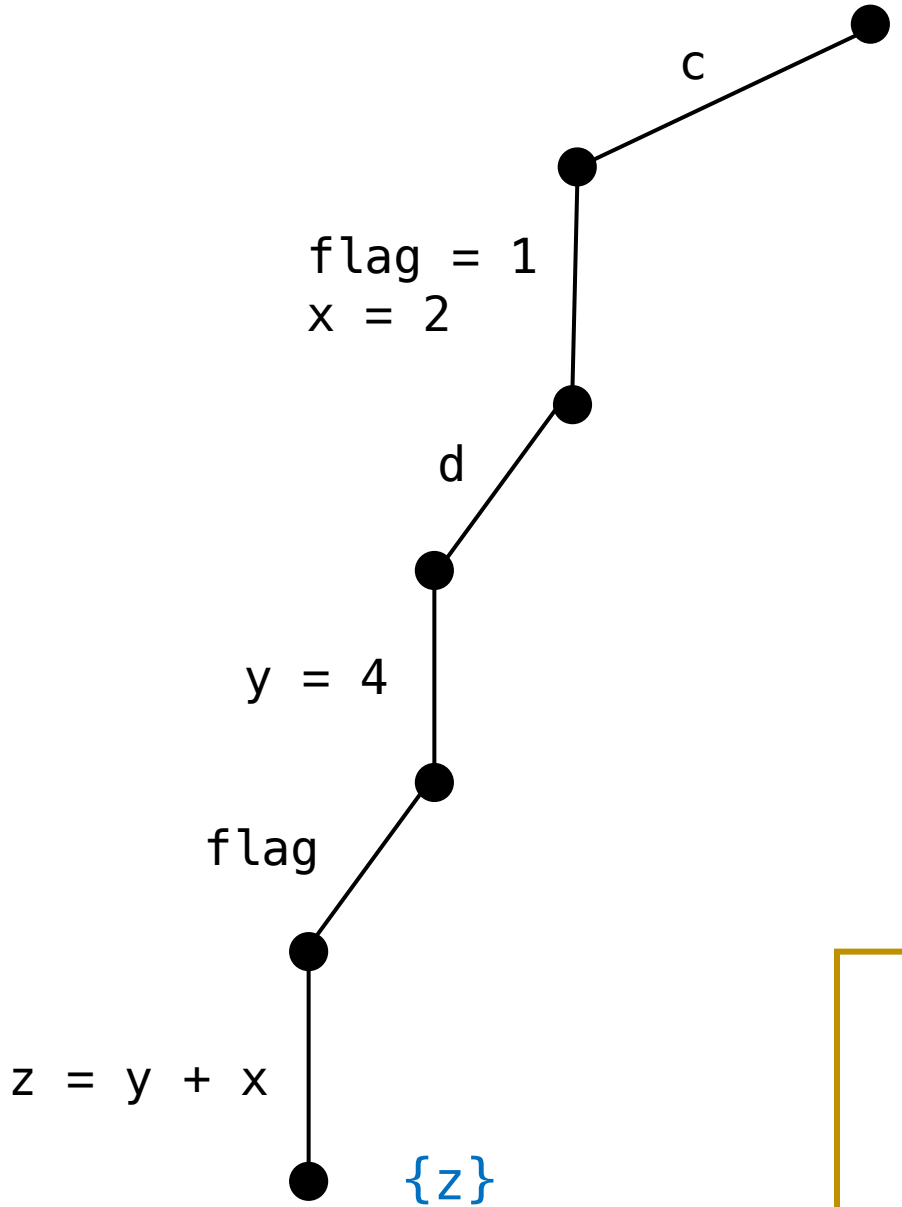



```
  c
∧ flag = 1
∧ x = 2
∧ d
∧ y = 4
∧ flag
∧ z = y + x
```

は満たされるので
このパスは実行可能

依存集合

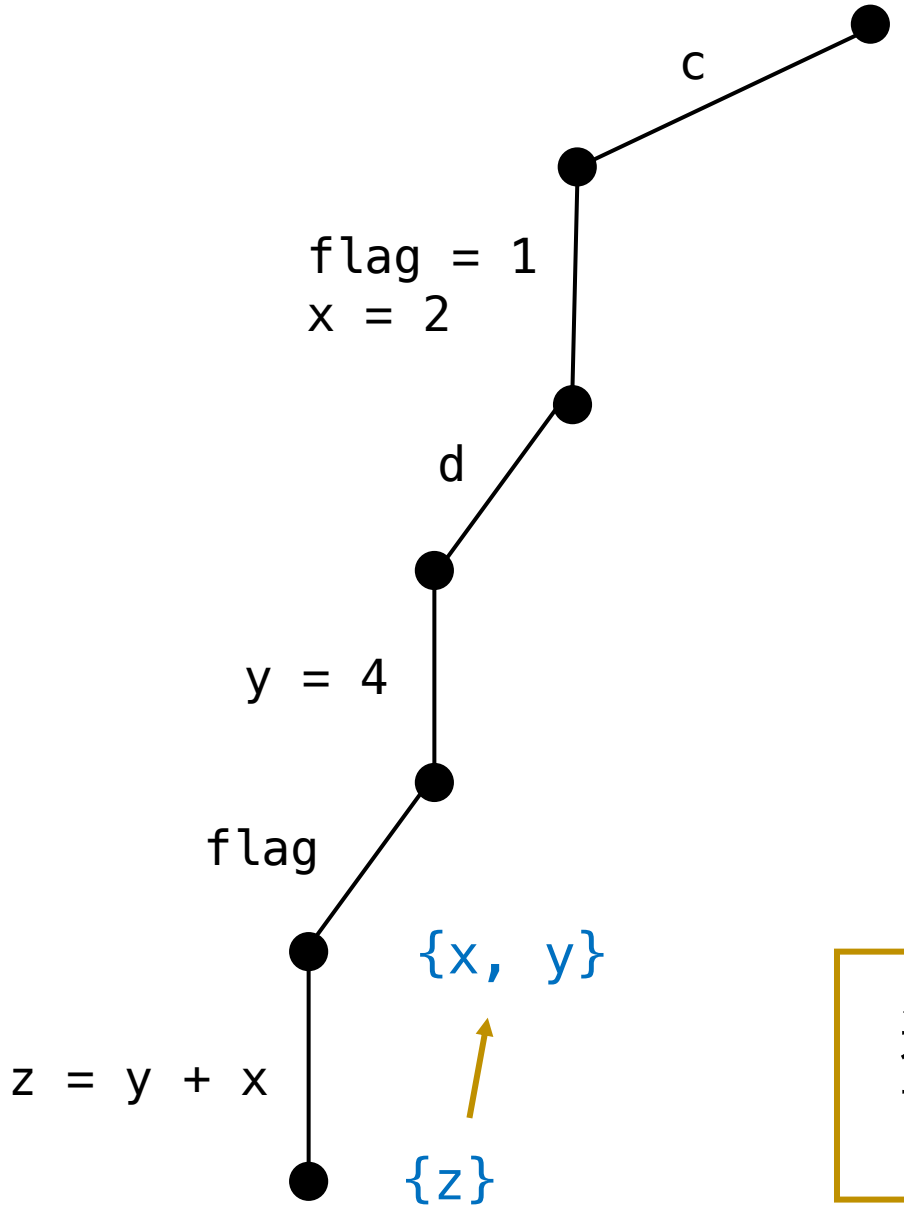
```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



実行可能なパスから
target 変数に関する
依存変数を計算

依存集合

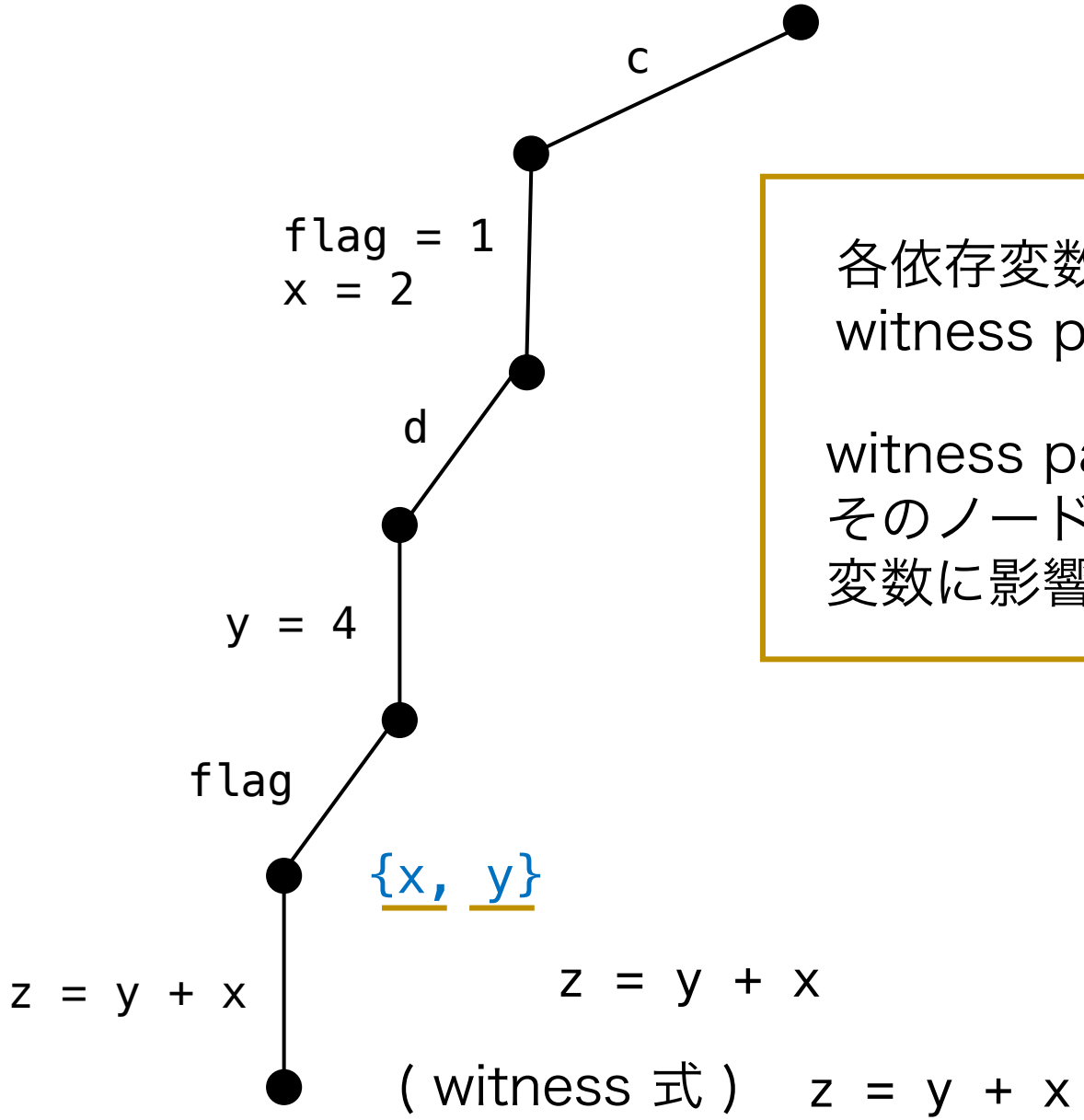
```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



逆向きに依存変数を計算

Witness Path

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

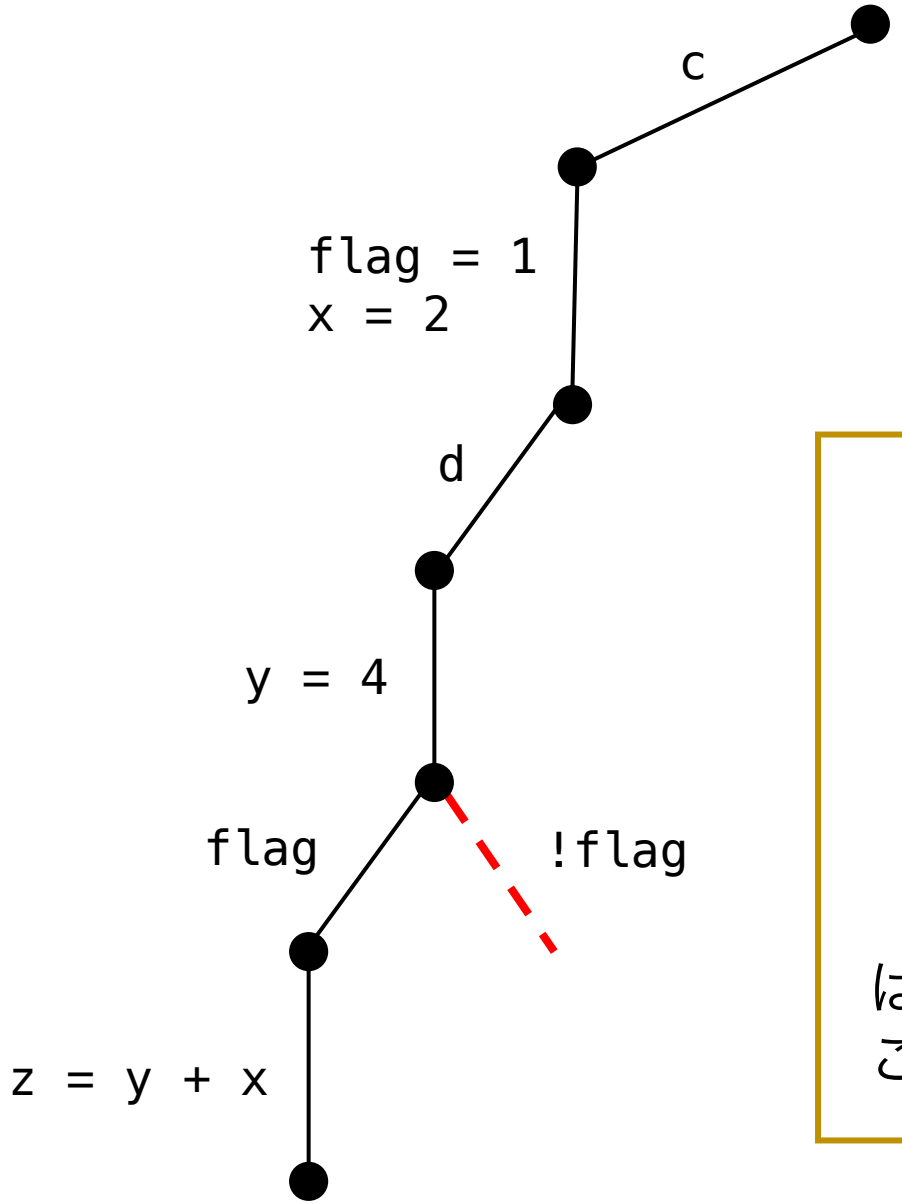


各依存変数に対して
witness path を計算

witness path :
そのノードから target
変数に影響を与えるパス

補間式

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

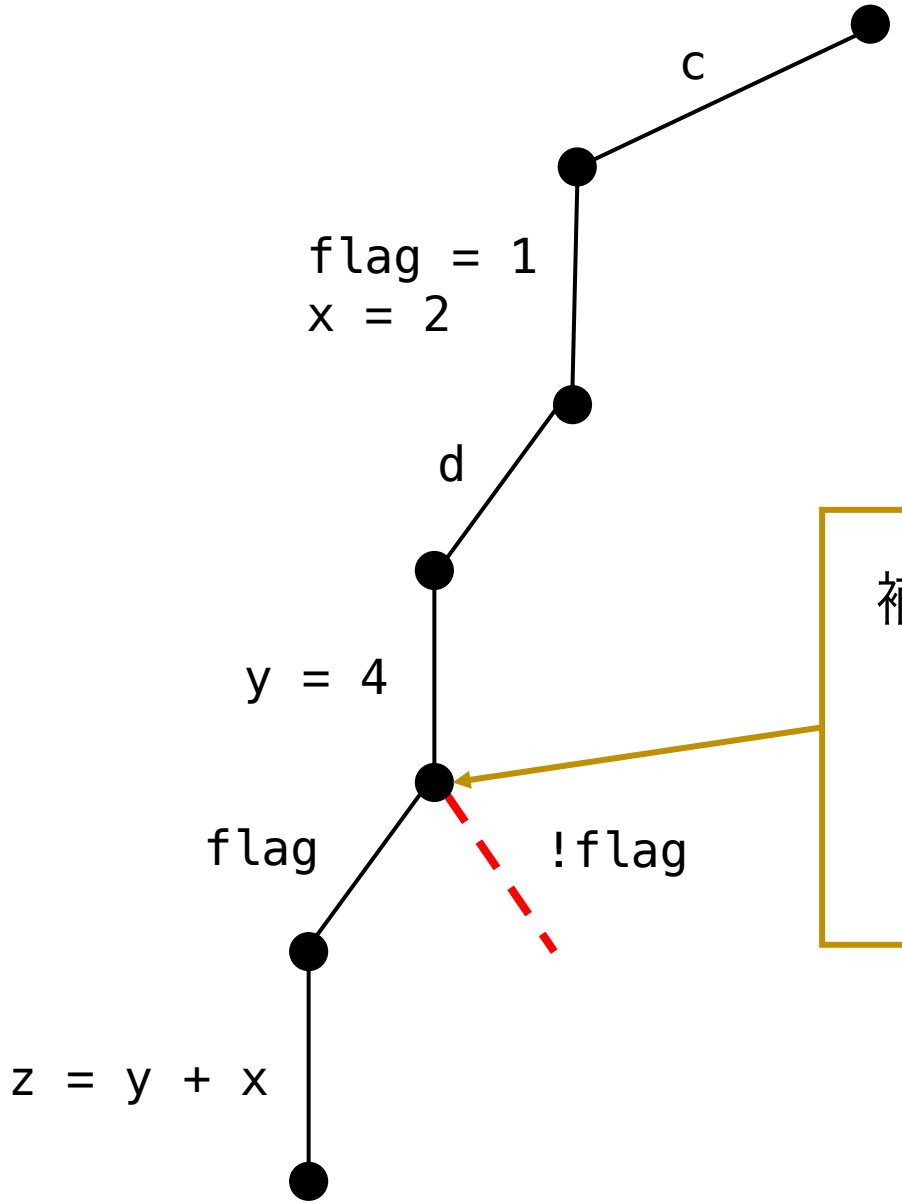


$$
\begin{aligned}
& c \\
\wedge\ & \texttt{flag} = 1 \\
\wedge\ & \texttt{x} = 2 \\
\wedge\ & \texttt{d} \\
\wedge\ & \texttt{y} = 4 \\
\wedge\ & \neg\, \texttt{flag}
\end{aligned}
$$

は満たされないので
このパスは 実行不可能

補間式

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



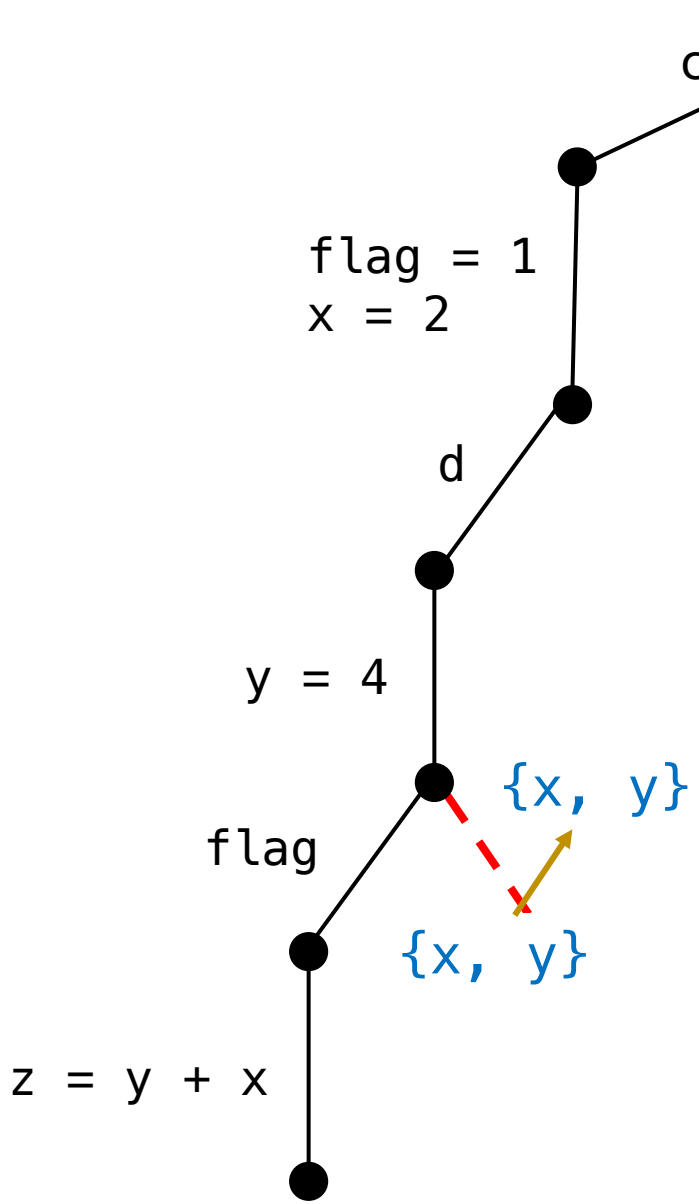
補間式：

実行不可能なパスが
満たさない性質

`flag = 1`

依存集合

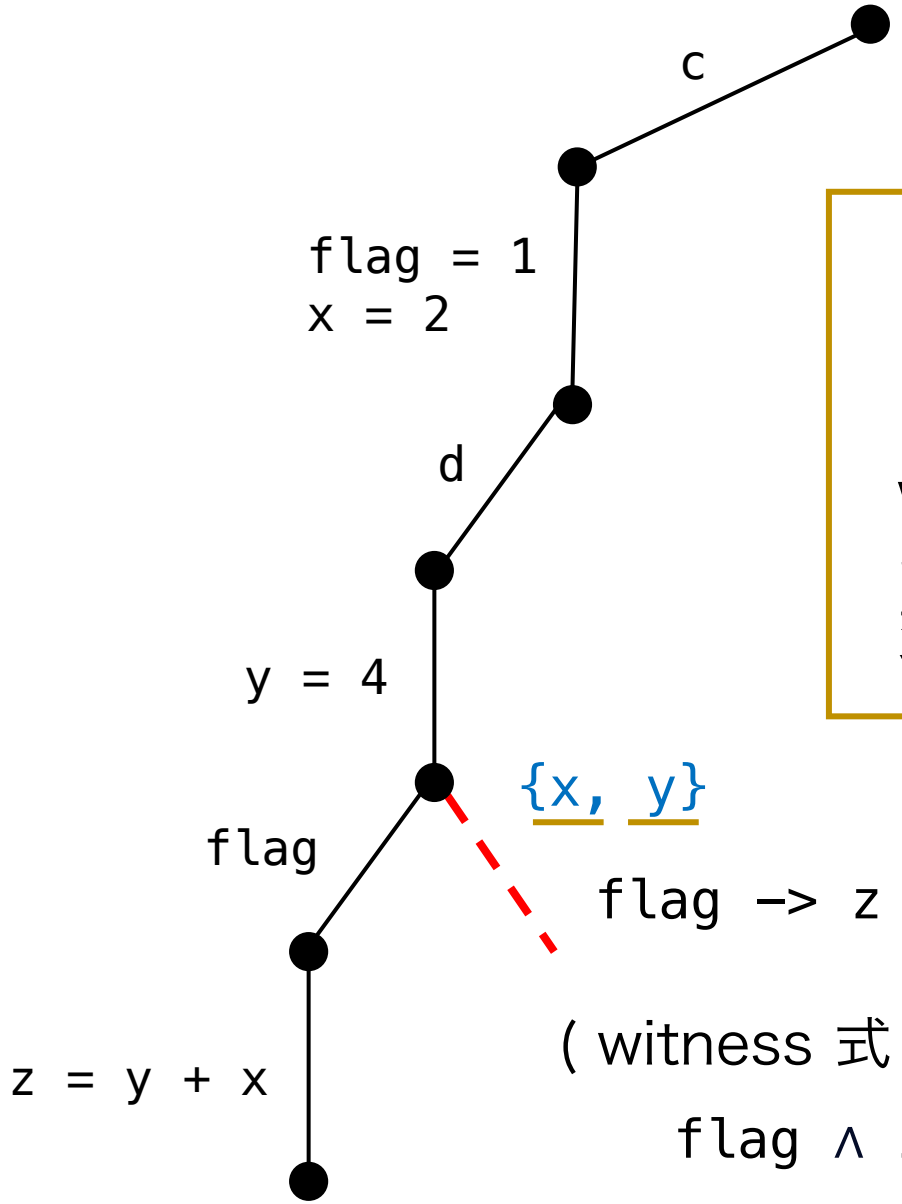
```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



逆向きに依存変数を計算

Witness Path

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

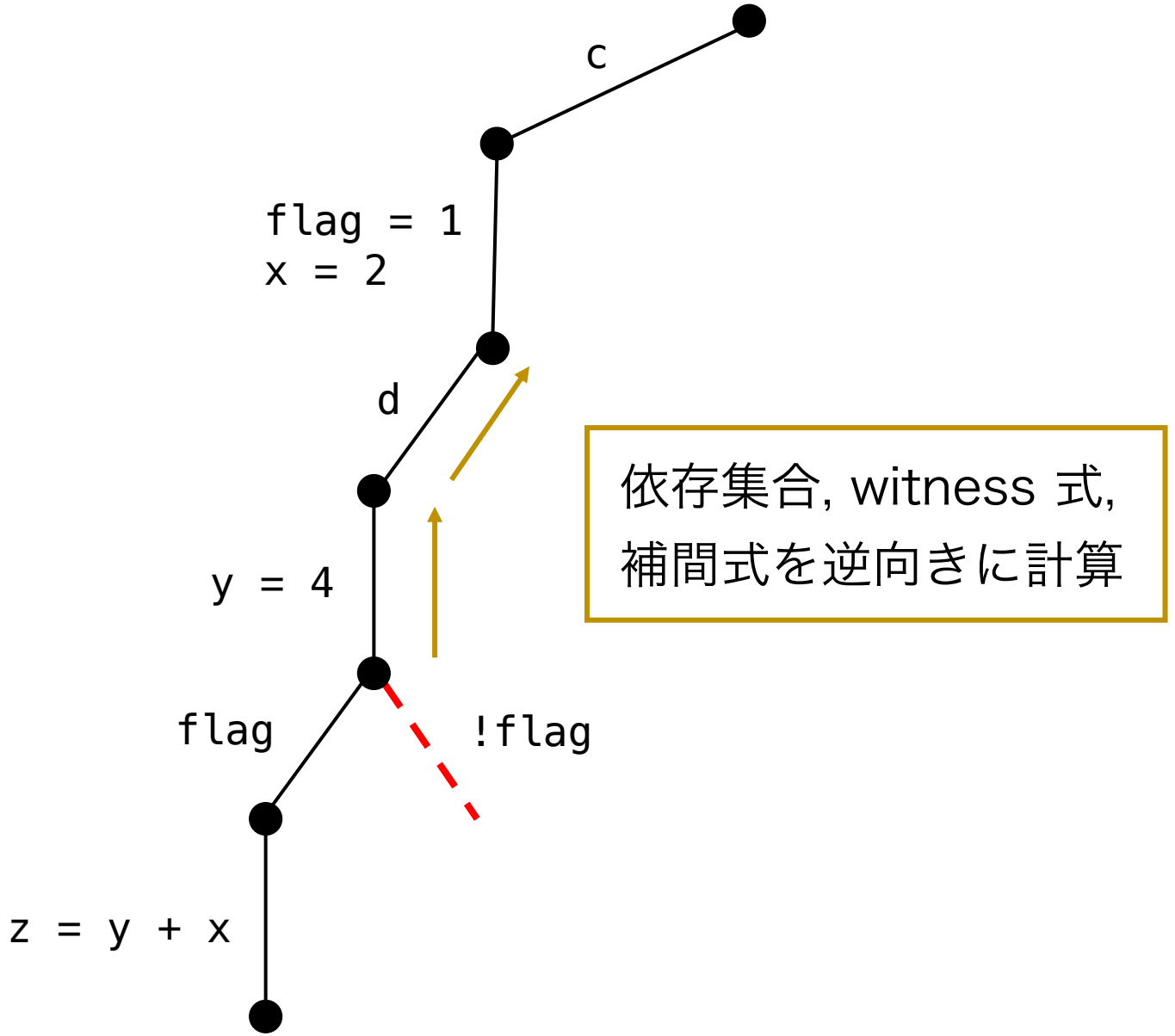


各依存変数に対して
witness path を計算

witness path :
そのノードから target
変数に影響を与えるパス

Backward

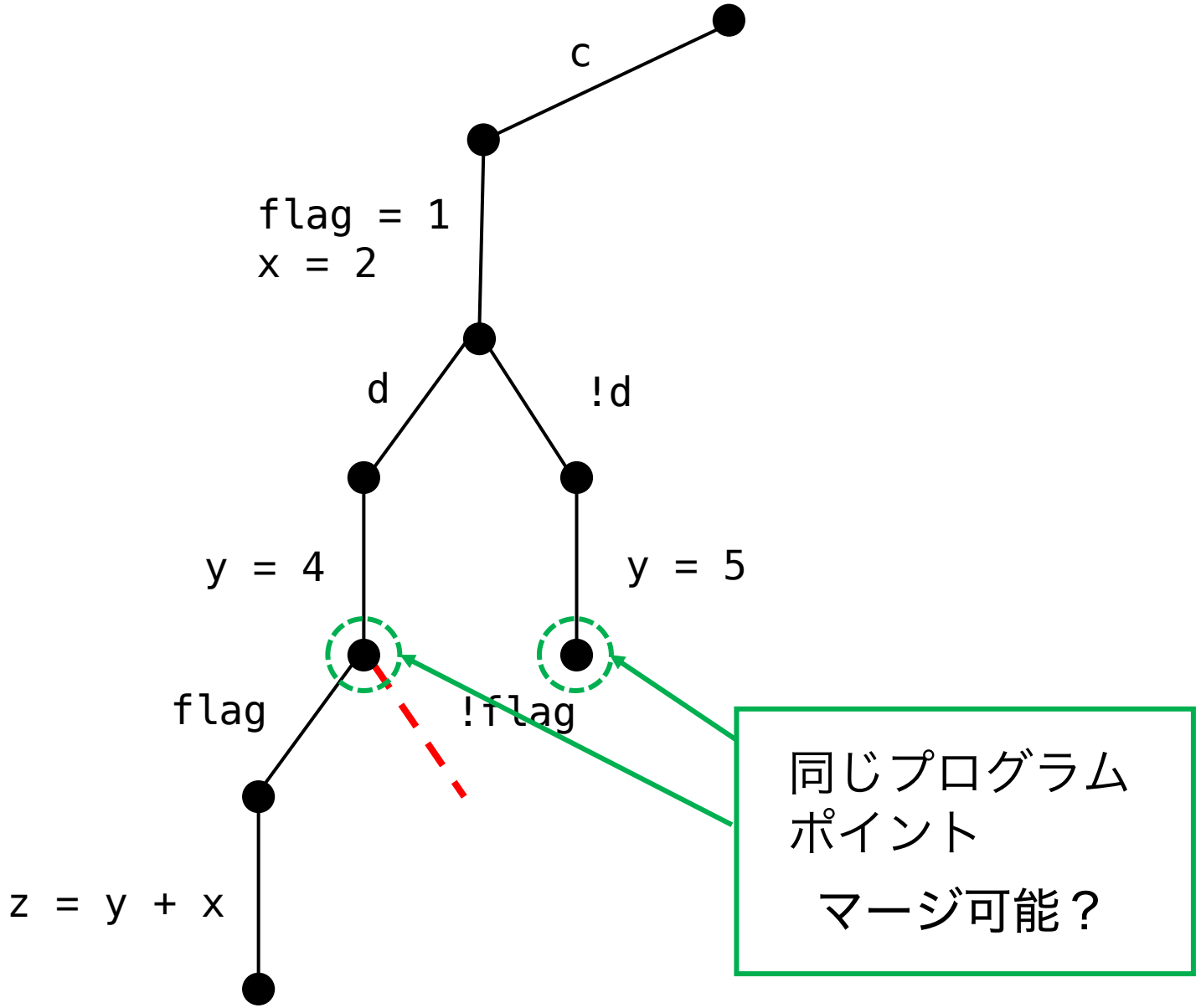
```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



依存集合, witness 式,
補間式を逆向きに計算

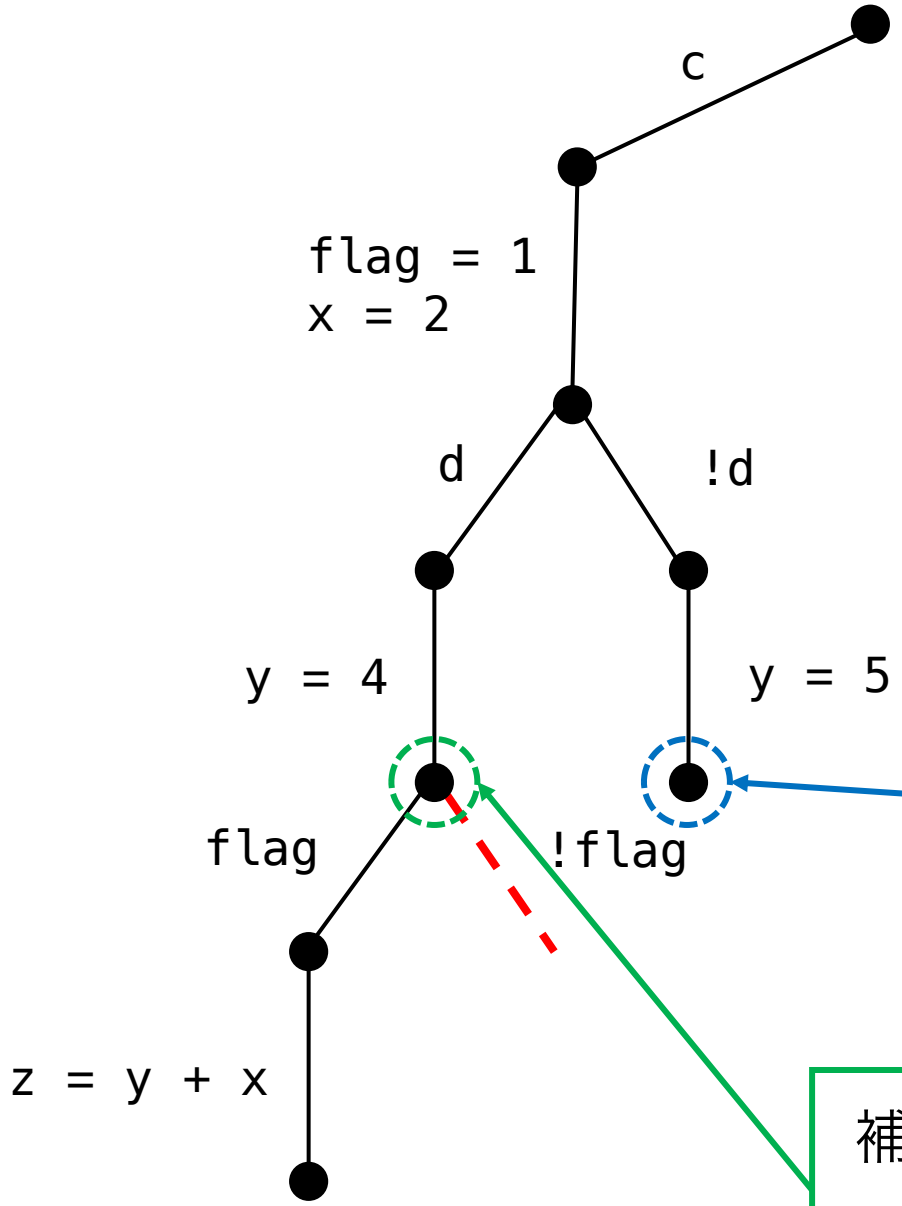
マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



1. 補間式に関する条件

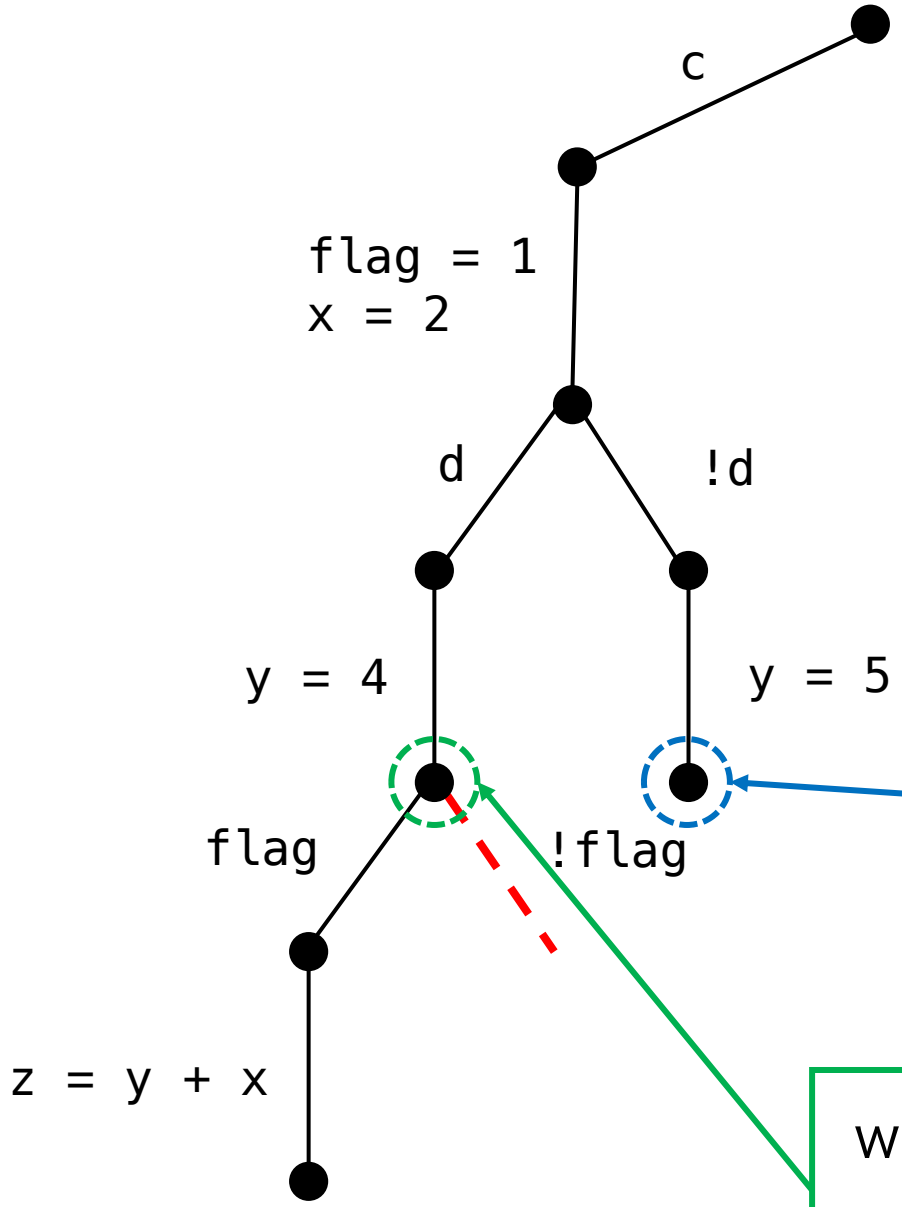
□ ⊨ □

c
∧ flag = 1
∧ x = 2
∧ ¬ d
∧ y = 5

補間式
flag = 1

マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



2. witness 式に関する条件

□ ∧ □

が充足可能

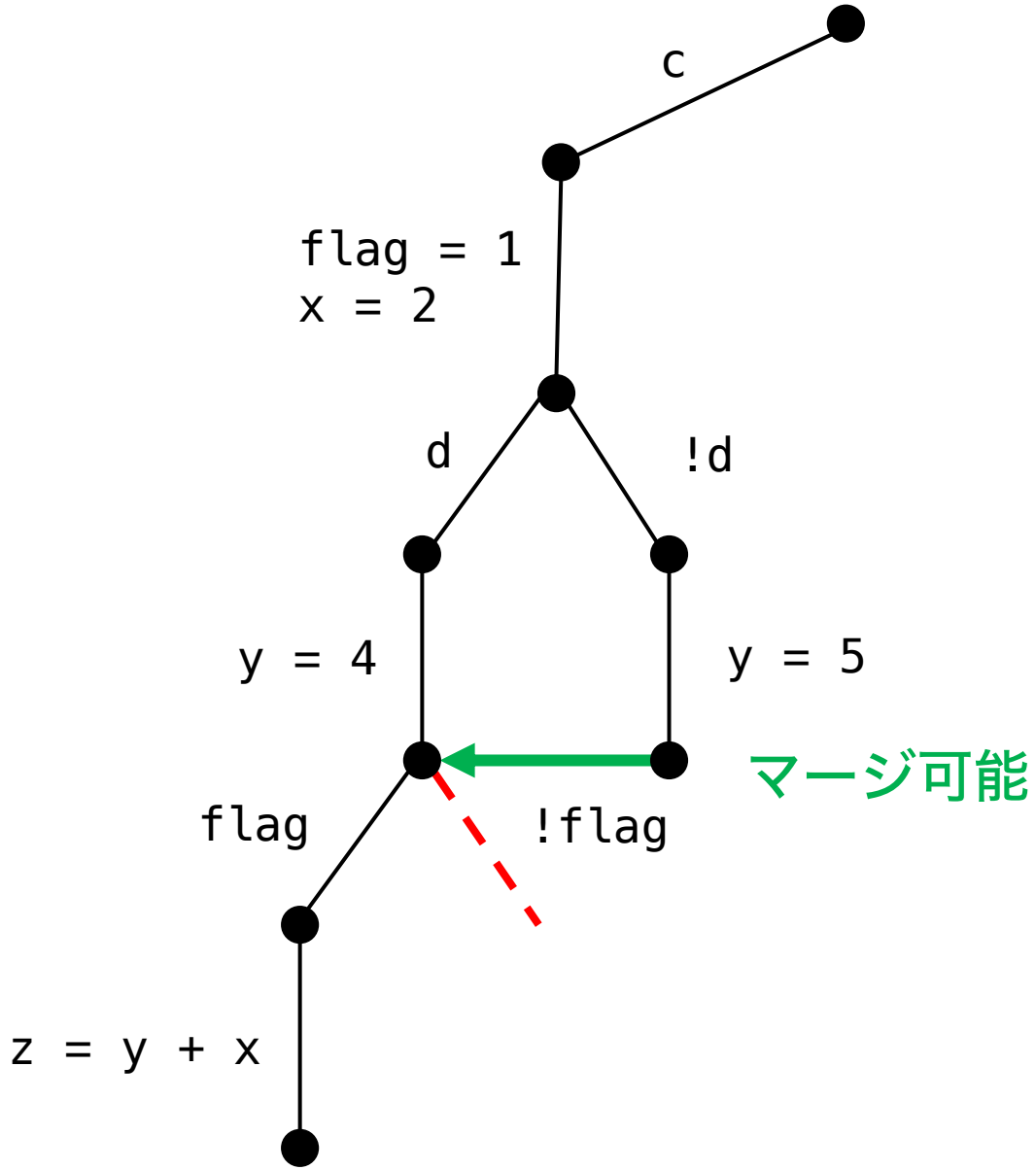
c
∧ flag = 1
∧ x = 2
∧ ¬ d
∧ y = 5

witness 式

flag ∧ z = y + x

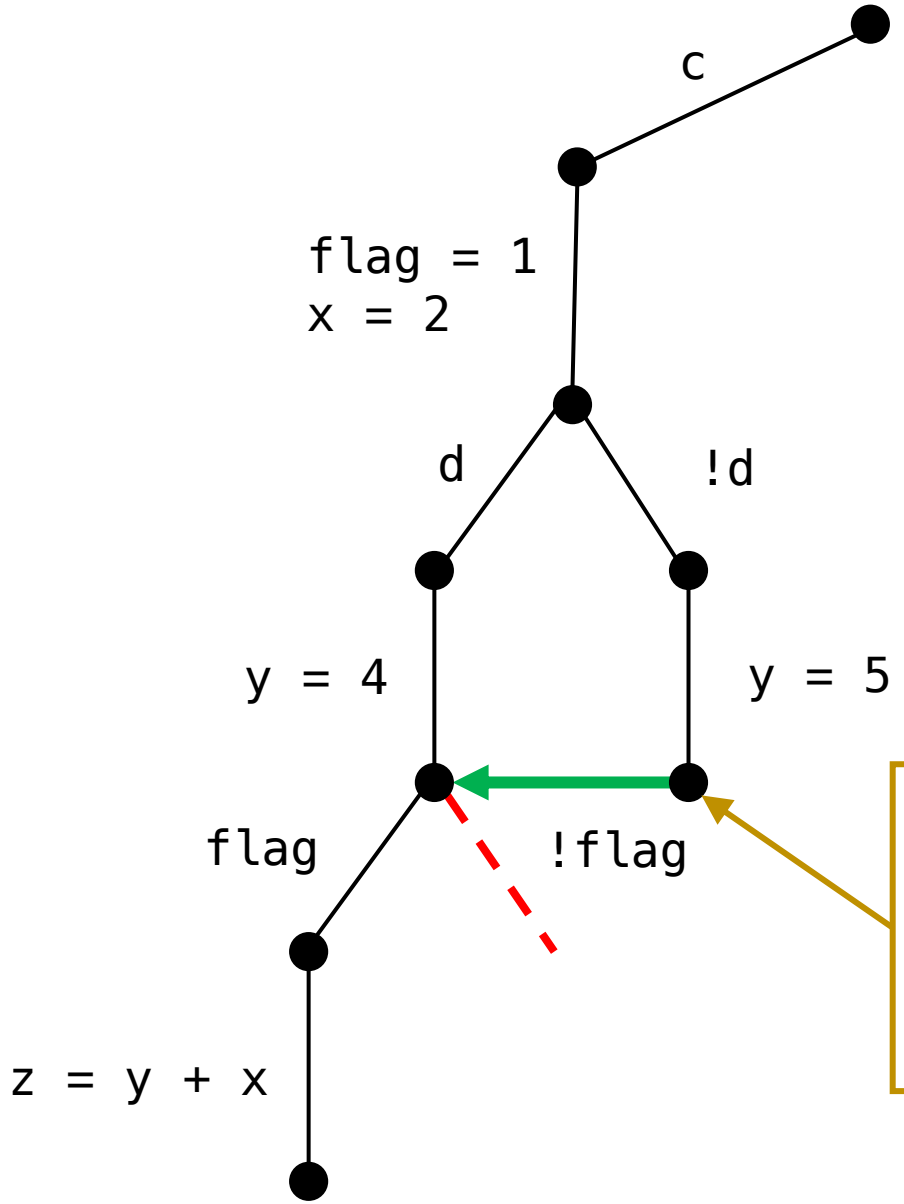
マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



マージ

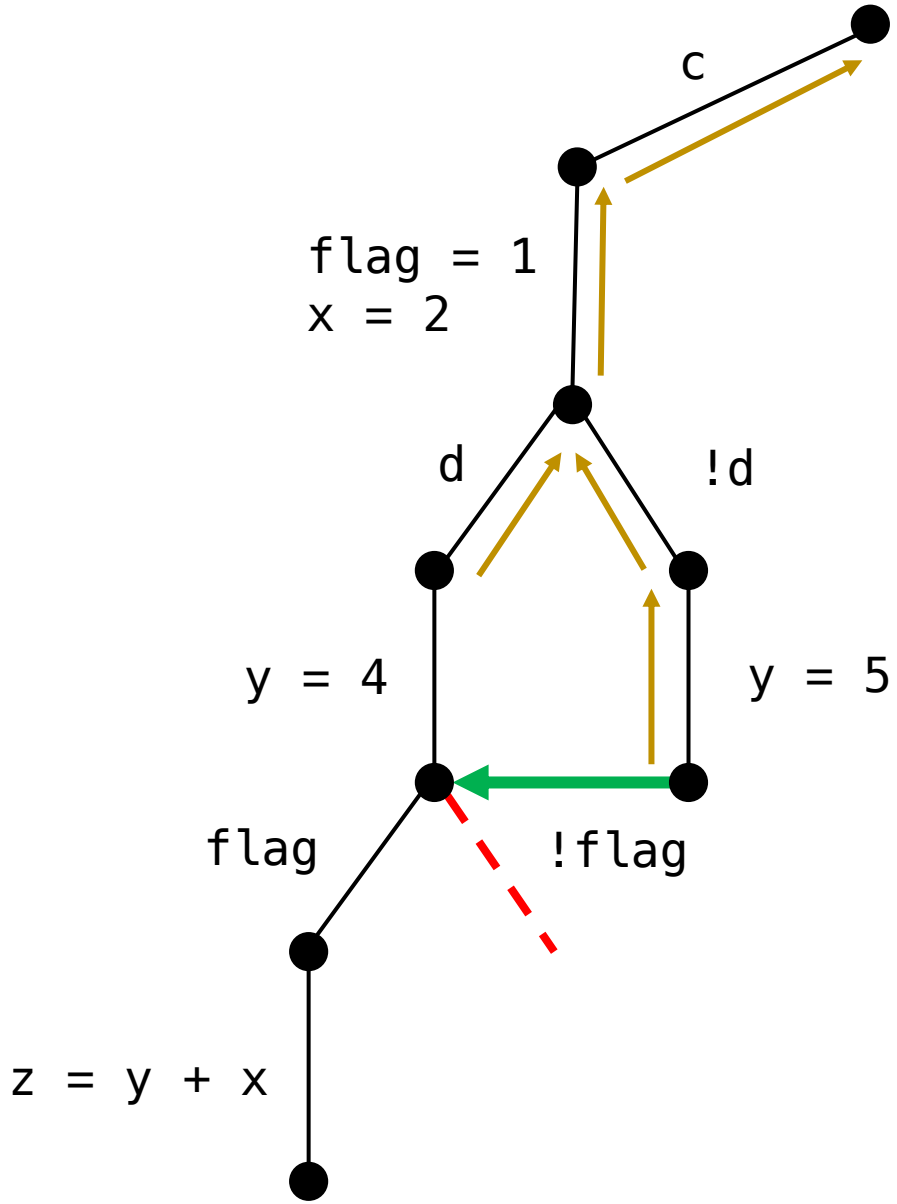
```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



依存集合, witness 式, 補間式をマージ先と同じにする

Backward

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



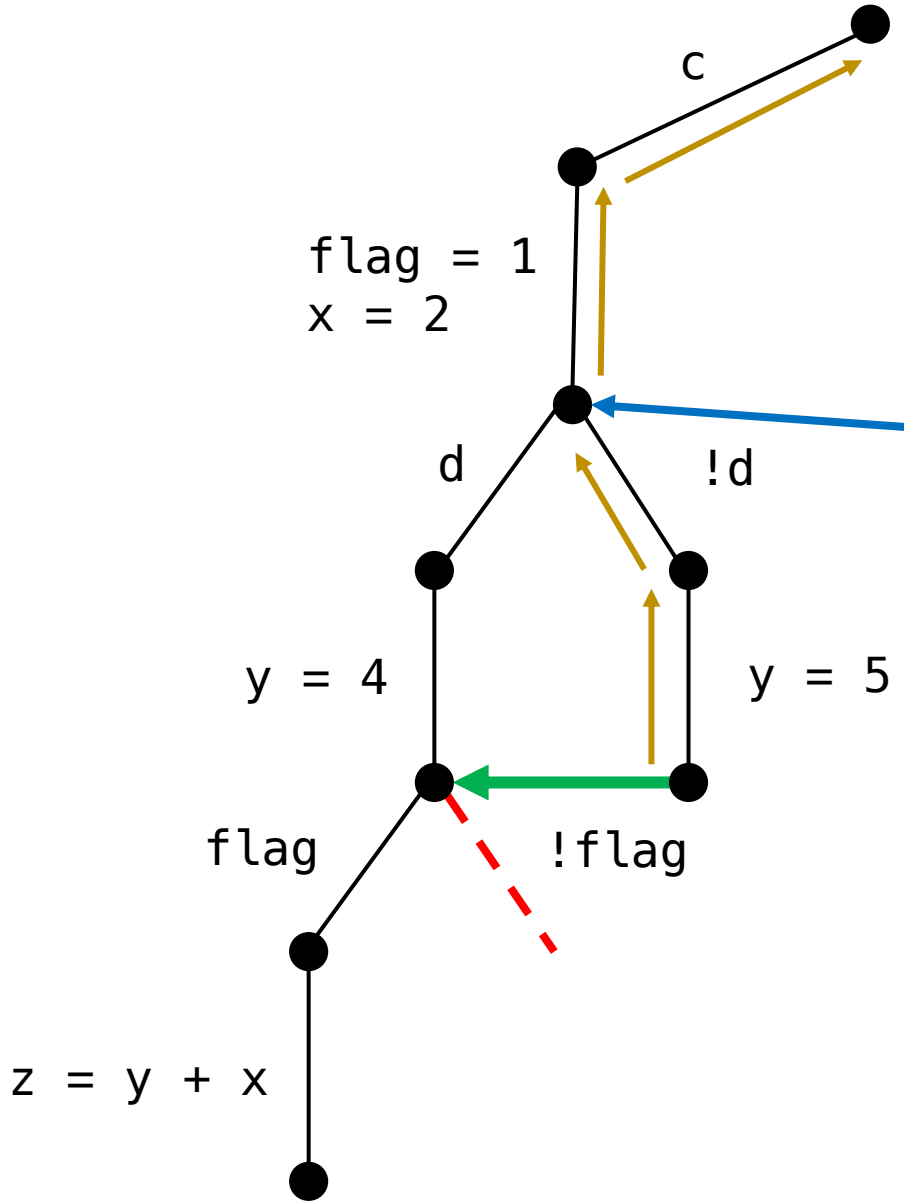
依存集合, witness 式, 補間式を逆向きに計算

依存集合, witness 式の結合は Union

補間式の結合は Conjunction

Backward

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



依存集合　{d, x}

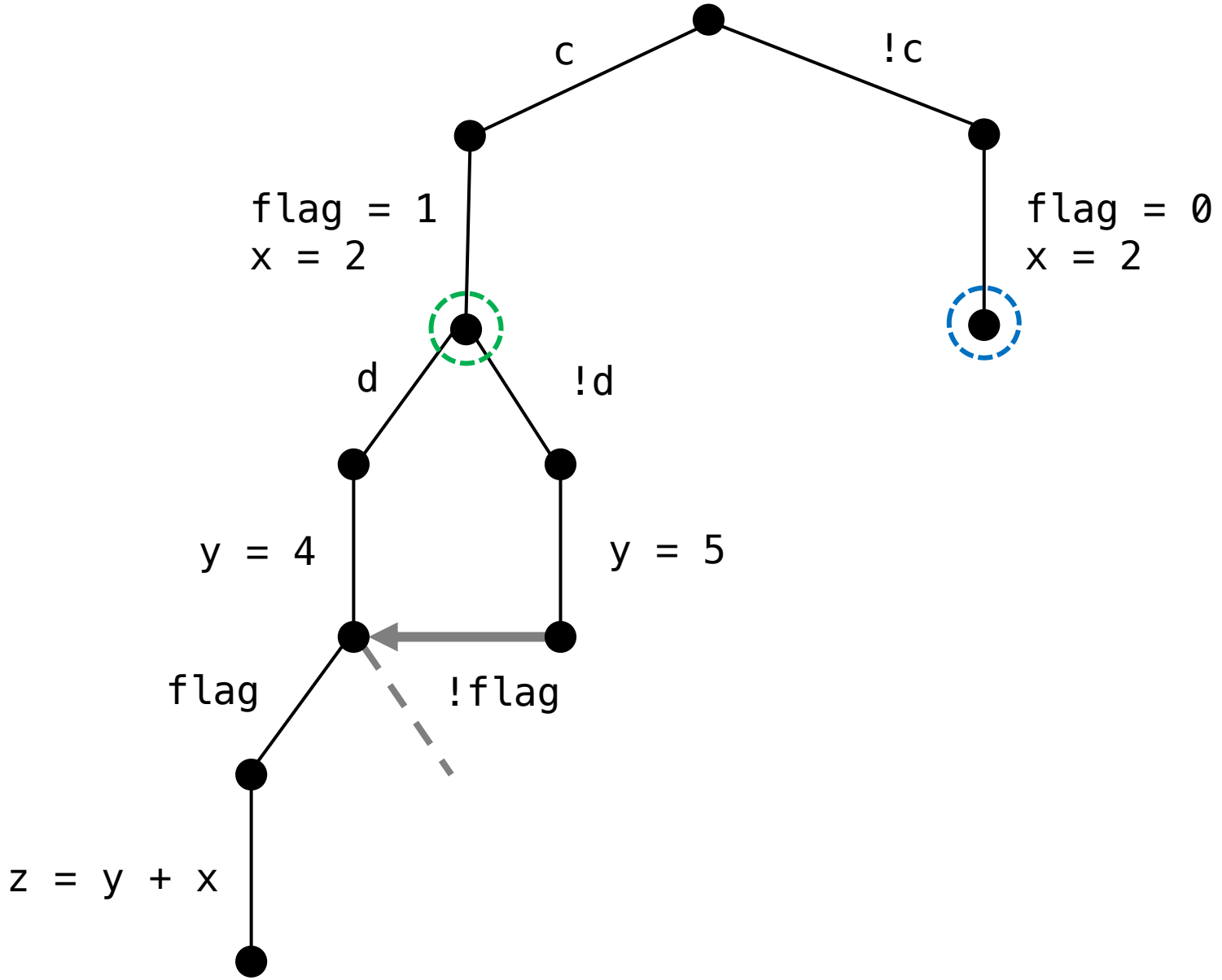
Witness 式

```
d, x:    d
       ∧ y = 4
       ∧ flag
       ∧ z = y + x
   or
         ¬d
       ∧ y = 5
       ∧ flag
       ∧ z = y + x
```

補間式　flag = 1

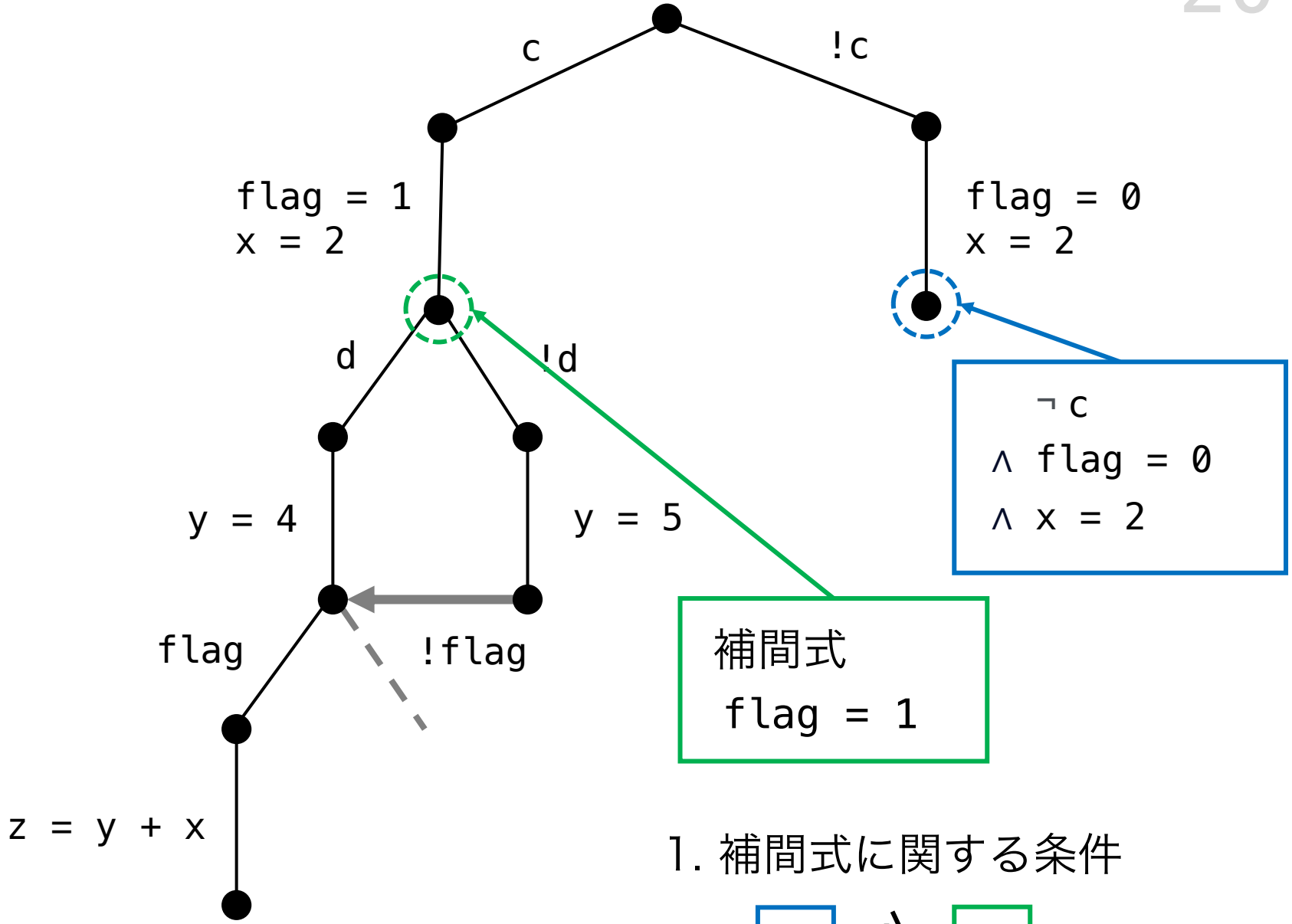
マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```

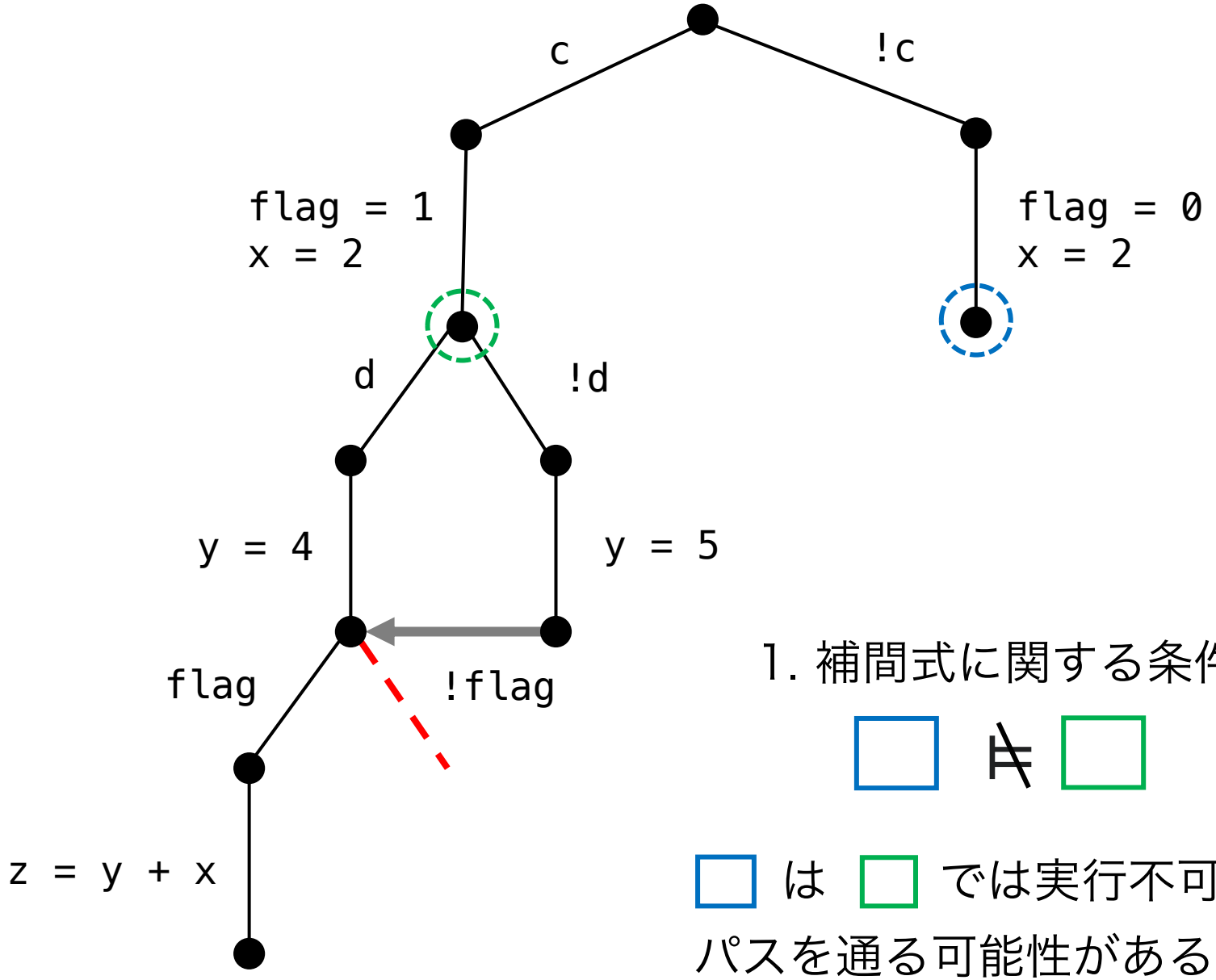


1. 補間式に関する条件

□ ⊭ □

マージ

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



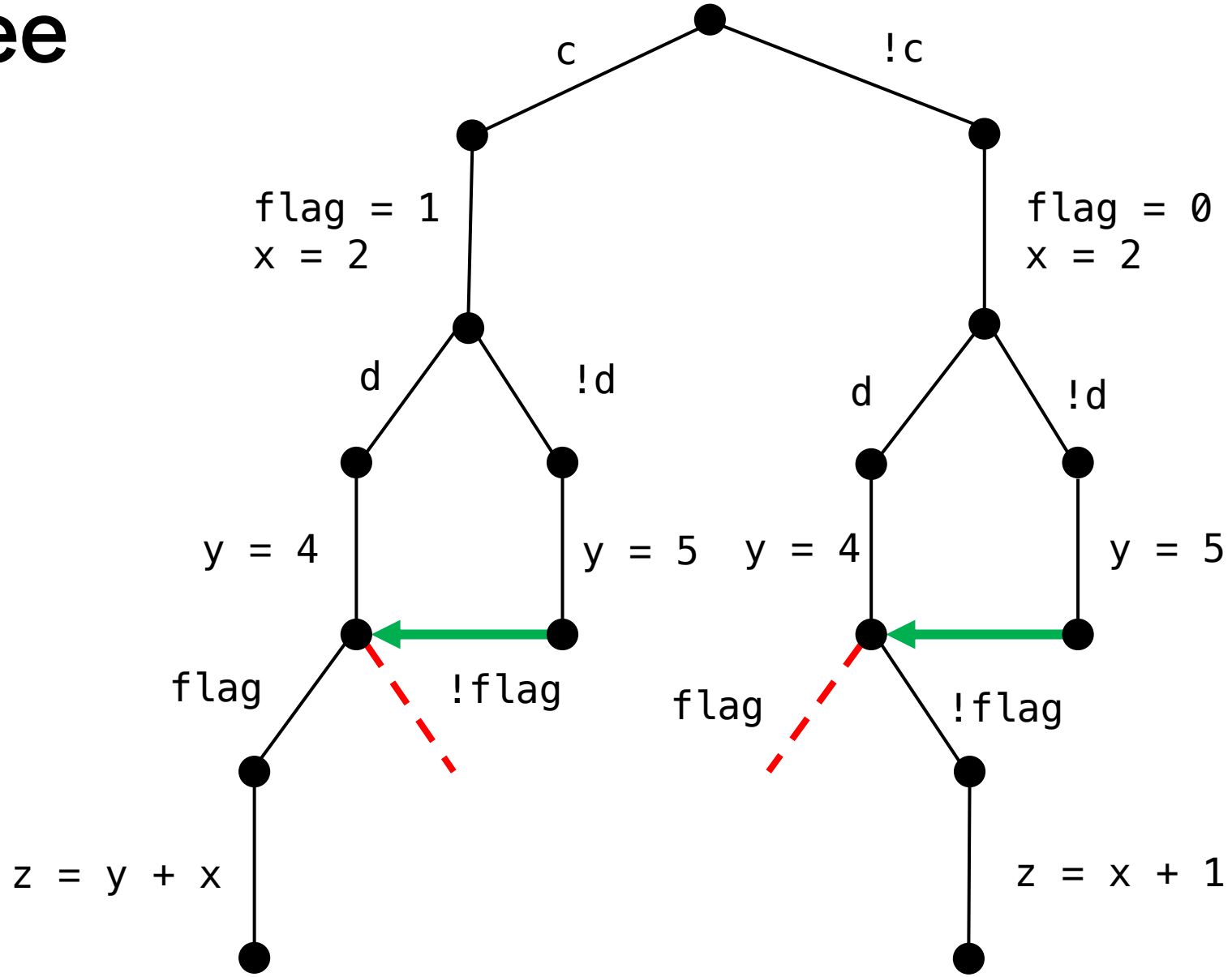
1. 補間式に関する条件

□ ⊭ □

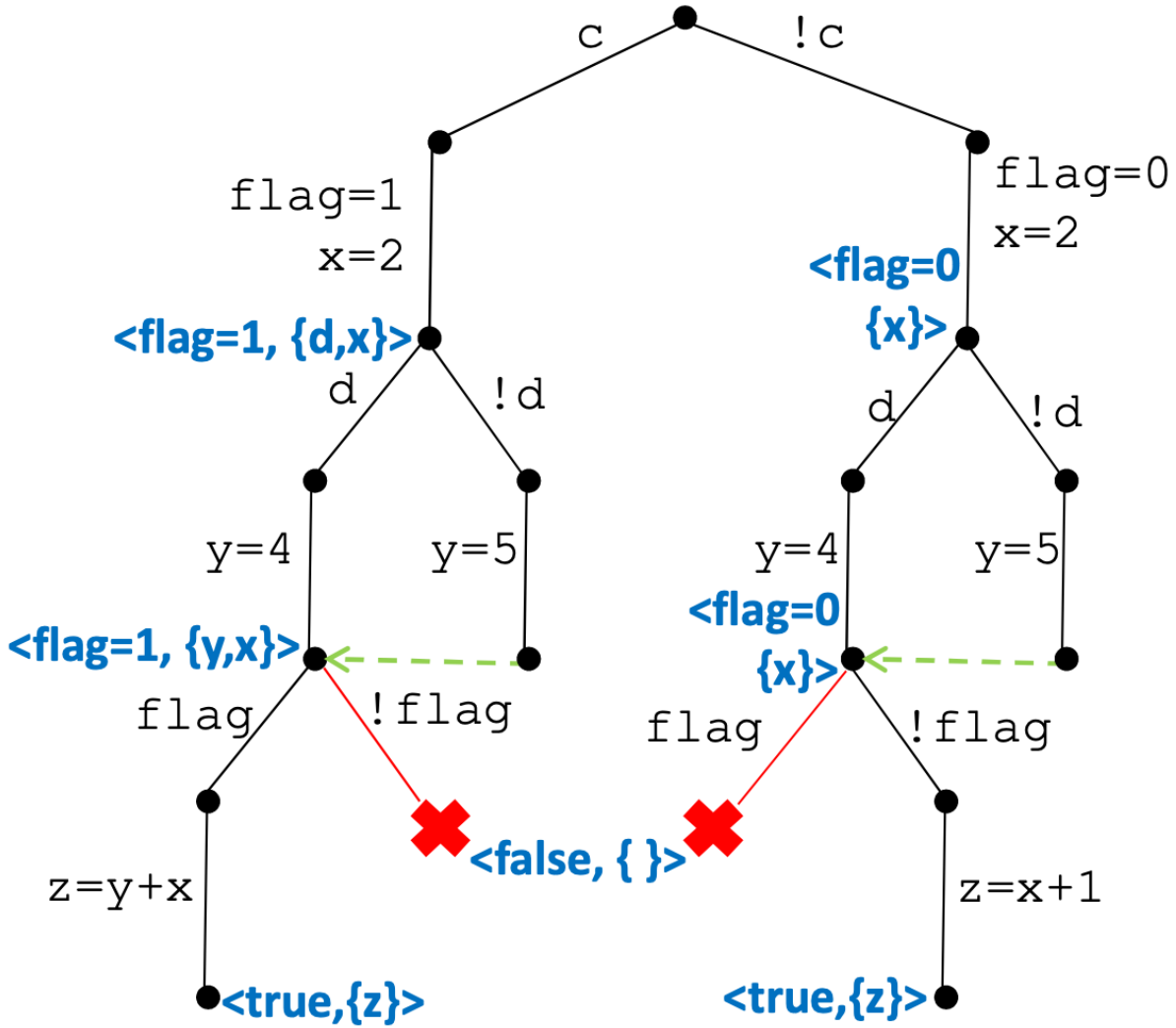
□ は □ では実行不可能な
パスを通る可能性がある

最終的な SE Tree

```
1   bool flag, c, d;
2   int x, y, z;
3
4   if (c) flag = 1;
5   else flag = 0;
6   x = 2;
7
8   if (d) y = 4;
9   else y = 5;
10
11  if (flag) z = y + x;
12  else z = x + 1;
13  target: {z}
```



最終的な SE Tree


c
!c
flag=1
x=2
flag=0
x=2
<flag=1, {d,x}>
<flag=0
{x}>
d
!d
d
!d
y=4
y=5
y=4
y=5
<flag=1, {y,x}>
<flag=0
{x}>
flag
!flag
flag
!flag
<false, { }>
z=y+x
z=x+1
<true,{z}>
<true,{z}>