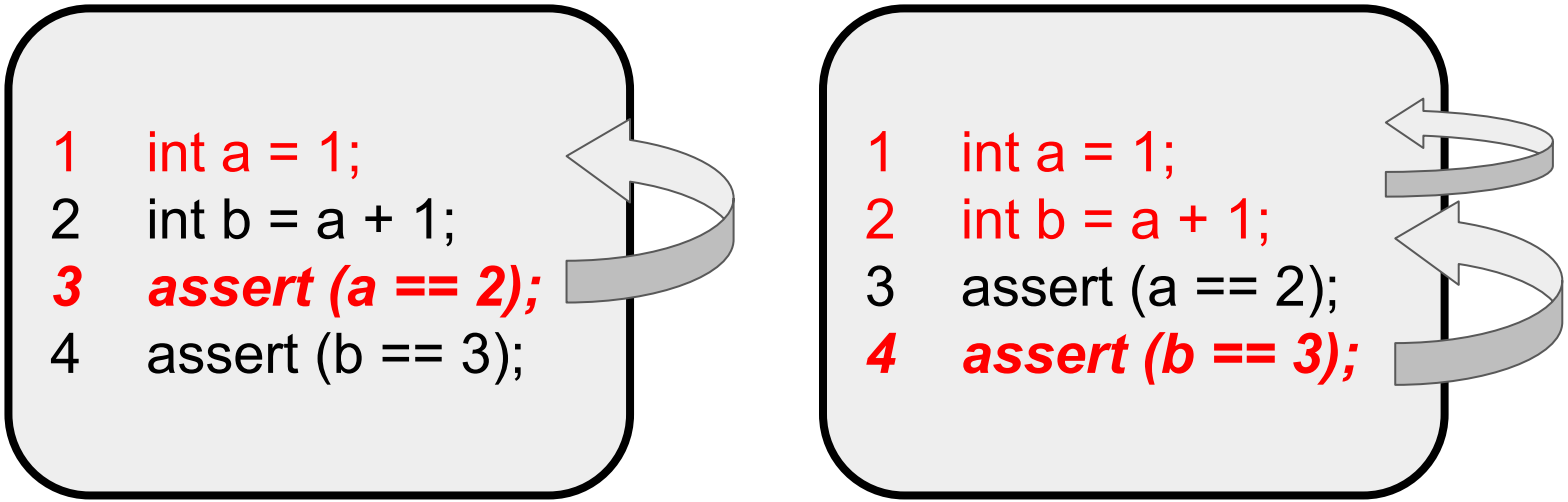
Program Slicing

What is slicing

Definition: The process that finding all statements in a program that directly or indirectly affect the value of a variable occurrence



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v_1, ..., v_n
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v_1) ... (c, v_n)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

Control Dependencies

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6            p = p - 1;
7    }
8    assert(p == 4);
```

If	A can alter the program's control and it determines whether B executes
Then	B is control-dependent on A

➢ 6 is control-dependent on 5

Example statements: ***if*** and ***while***

Control-flow Dependencies

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6            p = p - 1;
7    }
8    assert(p == 4);
```

If	B is executed immediately after A in the same execution thread
Then	B is control-flow-dependent on A

- 2 is control-flow-dependent on 1
- 3 is control-flow-dependent on 2
- 4 is control-flow-dependent on 3
- ……

Data-flow Dependencies

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6            p = p - 1;
7    }
8    assert(p == 4);
```

If	a). B is control-flow-dependent on A b). v used in B is defined in A and no other statement redefines v between A and B
Then	B is data-flow-dependent on A *i.e.* ***dynamic reaching definition*** *of v in B is A*

- 4 is data-flow-dependent on 1
  - Dynamic reaching definition of **m** in 4 is 1
- 5 is data-flow-dependent on 1
  - Dynamic reaching definition of **m** in 5 is 1
- 6 is data-flow-dependent on 3
  - Dynamic reaching definition of **p** in 6 is 3

Statement & Statement instance

s_i = **Statement**

The statement in line i

t_i^j = **Statement instance**

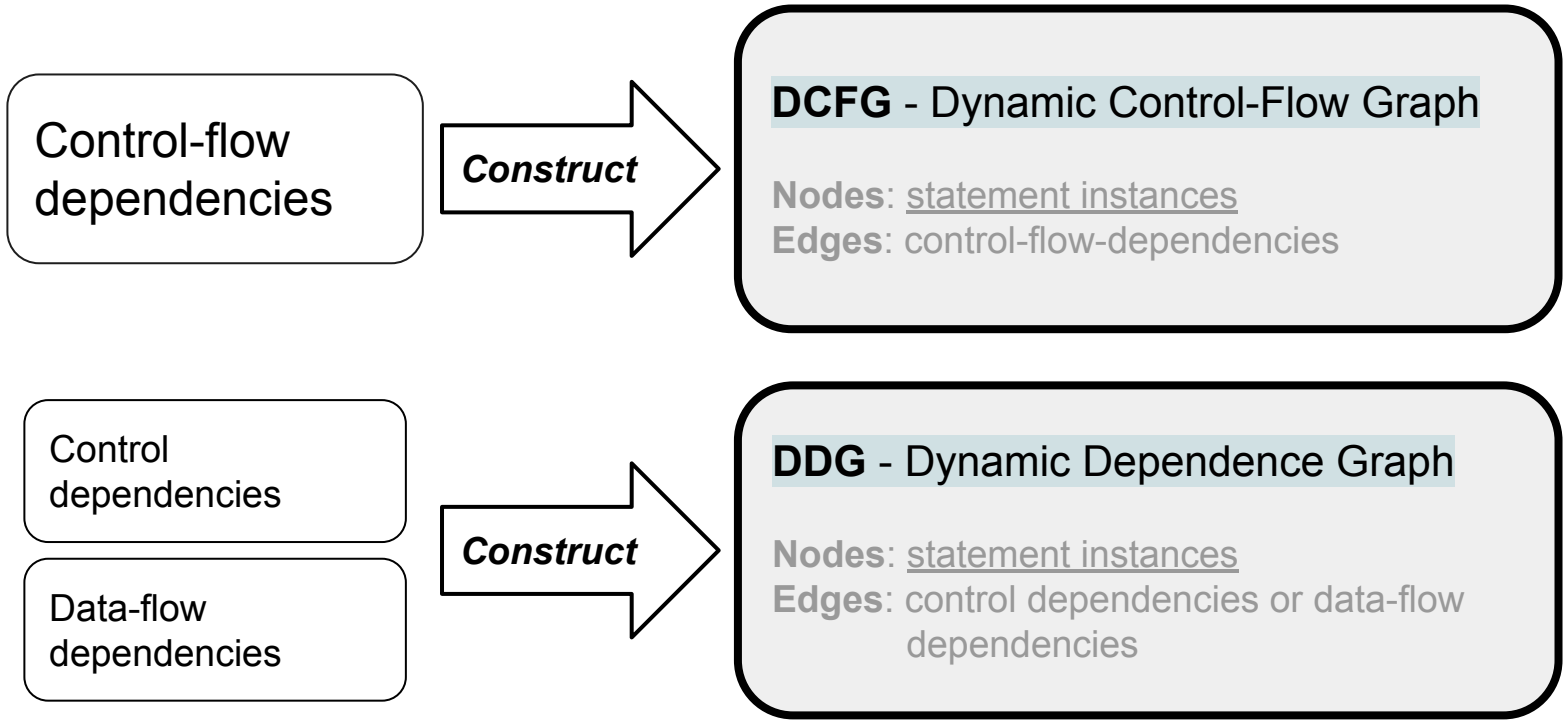
jth execution of statement s_i

```
1    int a = 1;
2    while (a >= 0) {
3        a = a - 1;
4    }
```

$$s_1, s_2, s_3$$

$$t_1^1, t_2^1, t_3^1, t_2^2, t_3^2, t_2^3$$

Graphs: DCFG and DDG

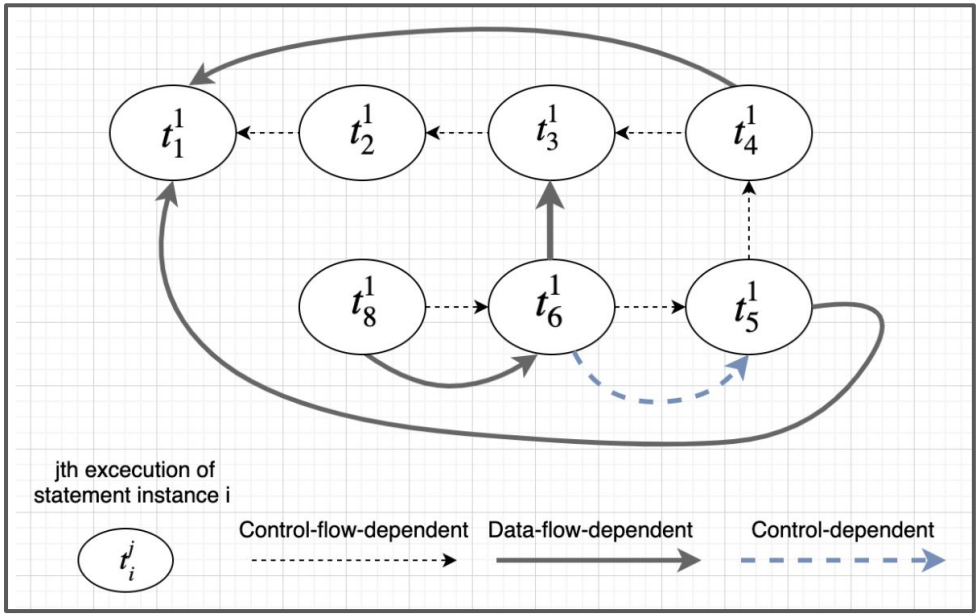


Graph: CDDG

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6          p = p - 1;
7    }
8    assert(p == 4);
```

CDDG - Combined Dynamic Dependence Graph

Combination of CDFG and DDG



Input: (c, v)

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6            p = p - 1;
7    }
8    assert(p == 4);
```

Slicing Criterion - (c, v)

A statement instance and all variables of interest **used** in this statement instance

Examples: $(t_4^1, m), (t_5^1, m), (t_6^1, p), (t_8^1, p)$

Algorithm 1: Dynamic slicing algorithm.

```
1  Input  : DCFG, c, v_1, ..., v_n
   Output: slice
2  begin
       ▷ Create                                   t instances and used
         variable
3      W ← {(c,
4      while W ≠
5          (t, v)
6          slice
7          t' ←
8          if t' n
               ▷
9              W                                  more than once
                                                  all its used variables
10         t' ←
11         if t' n
12             W
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```

$$c = t_8^1$$

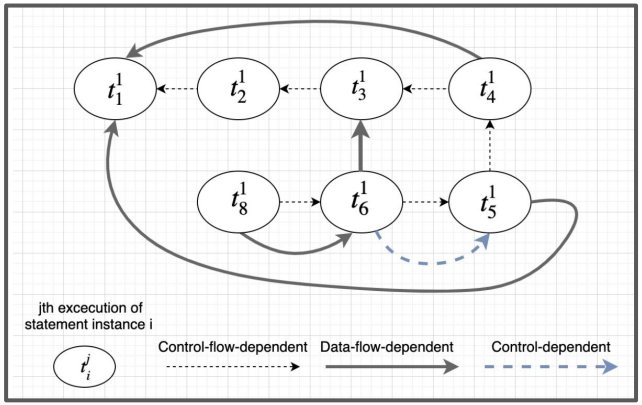
$$v_1 = p$$

Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          v) ← pick and remove from W
6          ce ← slice ∪ {t}
7          ← the instance t is control-dependent on
8          t′ not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t′, v′) | v′ ∈ use(t′)}
                       ▷ Add t′ to working set with all its used variables
10         ← reaching definition of v at t
11         t′ not in slice then
12             W ← W ∪ {(t′, v′) | v′ ∈ use(t)}
13     retu  slice
14 Procedu use(t)
15     retu  all variables used in t
```

$W \leftarrow \{(t_8^1, p)\}$ $\quad slice \leftarrow \{\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

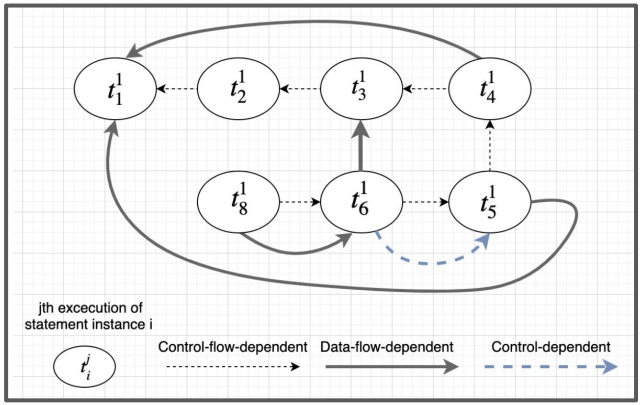
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instanc  is control-dependent on
8          if t' not in slice t
               ▷ Ensure not p  cessing statements more than once
9              W ← W ∪ {(t',  | v' ∈ use(t')}
                     ▷ Add t'  working set with all its used variables
10         t' ← reaching definition o  at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈  e(t)}
13     turn slice
14 Proc  ure use(t)
15     turn all variables used in t
```

$W \leftarrow \{\}$

$slice \leftarrow \{t_8^1\}$

$t \leftarrow t_8^1,\ v \leftarrow p$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

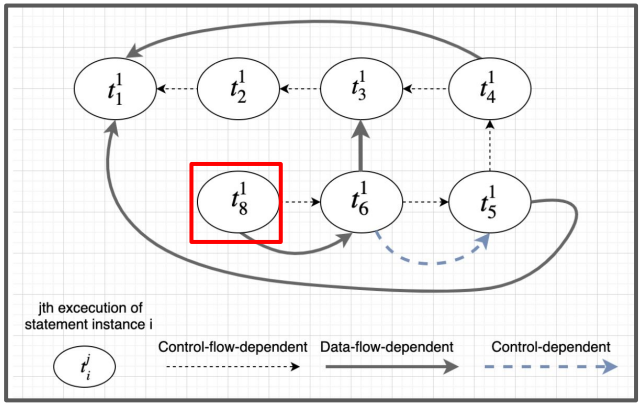
t' is empty (intra-method slicing)

Note: t' is the method containing t in inter-method slicing

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6          p = p - 1;
7    }
8    assert(p == 4);
```

$t \leftarrow t_8^1,\ v \leftarrow p$

$W \leftarrow \{\}$ $\quad slice \leftarrow \{t_8^1\}$



Algorithm 1: Dynamic slicing algorithm.

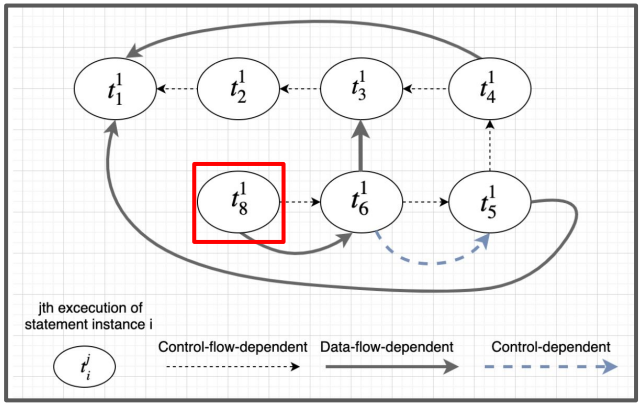
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t′ ← the instance t is control-dependent on
8          if t′ not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t′, v′) | v′ ∈ use(t′)}
                       ▷ Add t′ to working set with all its used variables
10         t′ ← reaching definition of v at t
11         if t′ not in slice then
12             W ← W ∪ {(t′, v′) | v′ ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t' \leftarrow t_6^1$

$t \leftarrow t_8^1,\ v \leftarrow p$

$W \leftarrow \{\}$ $\qquad$ $slice \leftarrow \{t_8^1\}$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6          p = p - 1;
7    }
8    assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

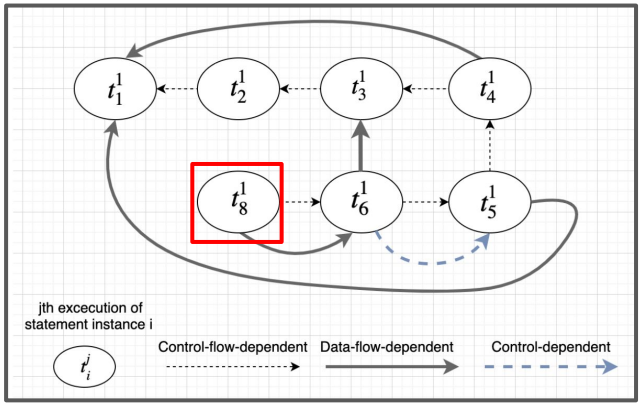
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t' \leftarrow t_6^1$

$t \leftarrow t_8^1,\ v \leftarrow p$

$W \leftarrow \{\}$ $\qquad$ $slice \leftarrow \{t_8^1\}$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```



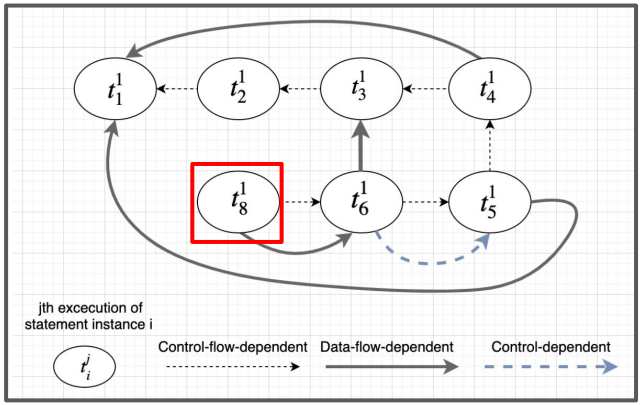
Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slic
14 Procedure use
15     return all   riables used in t
```

$t \leftarrow t_8^1,\ v \leftarrow p$

$W \leftarrow \{(t_6^1, p)\}$ $\quad slice \leftarrow \{t_8^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

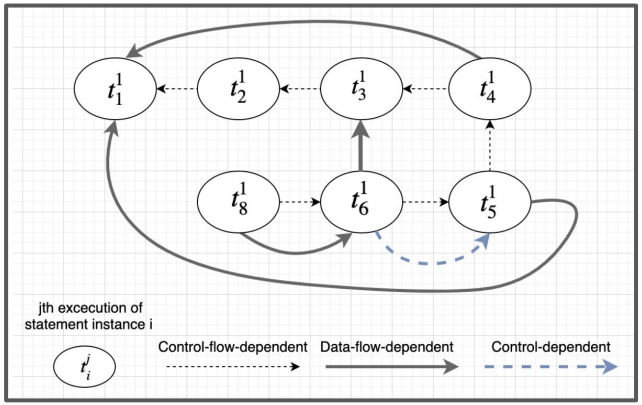
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure no processing statements more than once
9              W ← W ∪ {(t, v') | v' ∈ use(t')}
                   ▷ Add t to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t \leftarrow t_6^1,\ v \leftarrow p$

$W \leftarrow \{\}$

$slice \leftarrow \{t_8^1, t_6^1\}$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

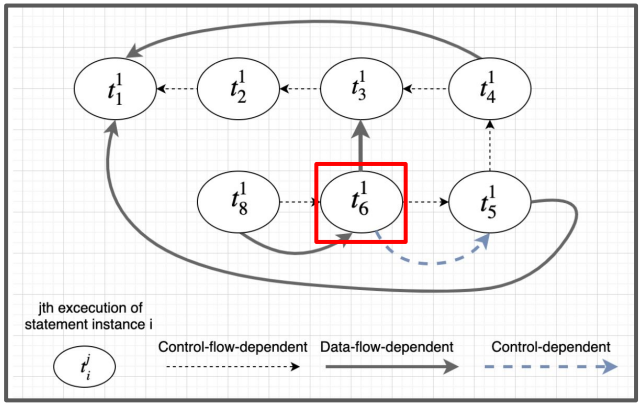
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t′ ← the instance t is control-dependent on
8          if t′ not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t′, v′) | v′ ∈ use(t′)}
                       ▷ Add t′ to working set with all its used variables
10         t′ ← reaching definition of v at t
11         if t′ not in slice then
12             W ← W ∪ {(t′, v′) | v′ ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t' \leftarrow t_5^1$

$t \leftarrow t_6^1,\ v \leftarrow p$

$W \leftarrow \{\}$ $\qquad$ $slice \leftarrow \{t_8^1,\ t_6^1\}$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t′ ← the instance t is control-dependent on
8          if t′ not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t′, v′) | v′ ∈ use(t′)}
                   ▷ Add t′ to working set with all its used variables
10         t′ ← reaching definition of t
11         if t′ not in slice then
12             W ← W ∪ {(t′, v′) | v′ ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

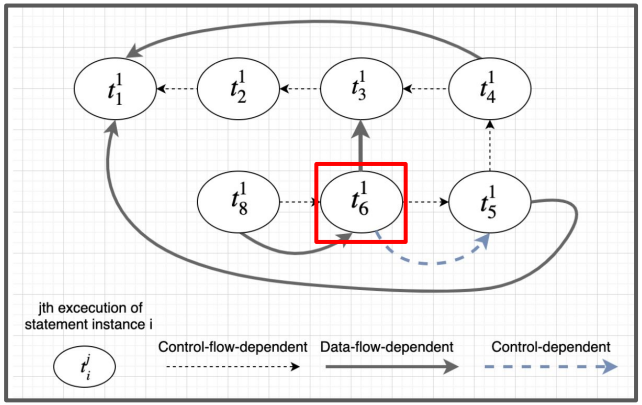
$t' \leftarrow t_5^1$

$t \leftarrow t_6^1,\ v \leftarrow p$

$W \leftarrow \{\}$

$slice \leftarrow \{t_8^1,\ t_6^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6       p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

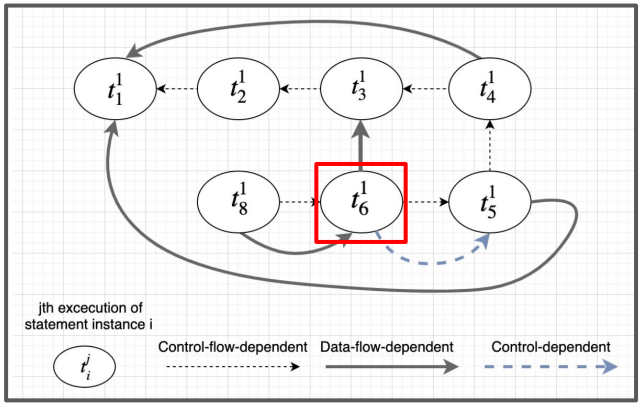
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t′ ← the instance t is control-dependent on
8          if t′ not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t′, v′) | v′ ∈ use(t′)}
                       ▷ Add t′ to working set with all its used variables
10         t′ ← reaching definition of v at t
11         if t′  t in slice then
12             ← W ∪ {(t′, v′) | v′ ∈ use(t)}
13     return sl
14 Procedure us  t)
15     return al  ariables used in t
```

$t \leftarrow t_6^1,\ v \leftarrow p$

$W \leftarrow \{(t_5^1, m)\}$

$slice \leftarrow \{t_8^1, t_6^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6       p = p - 1;
7   }
8   assert(p == 4);
```



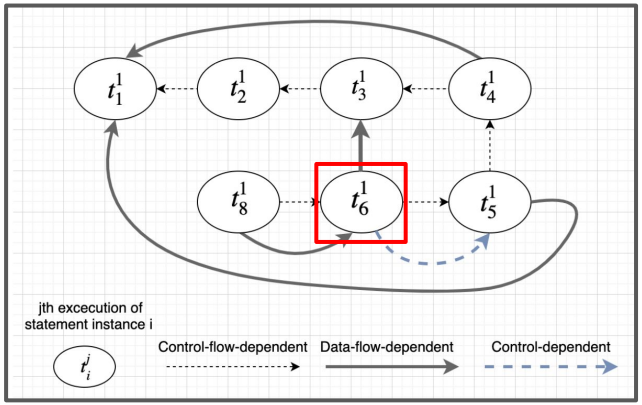
Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure us t)
15     return al ariables used in t
```

$t \leftarrow t_6^1,\ v \leftarrow p$

$W \leftarrow \{(t_5^1, m), (t_3^1, p)\}$ $\quad slice \leftarrow \{t_8^1, t_6^1\}$

```
1    int m = 1;
2    int n = 2;
3    int p = 4;
4    int q = m + 2;
5    if (m == 1) {
6        p = p - 1;
7    }
8    assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

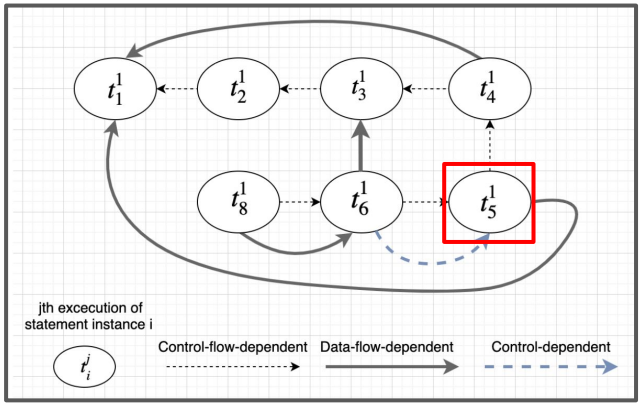
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure no processing statements more than once
9              W ← W ∪ {(t, v') | v' ∈ use(t')}
                   ▷ Add t to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$W \leftarrow \{(t_3^1, p)\}$

$t \leftarrow t_5^1,\ v \leftarrow m$

$slice \leftarrow \{t_8^1, t_6^1, t_5^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6       p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

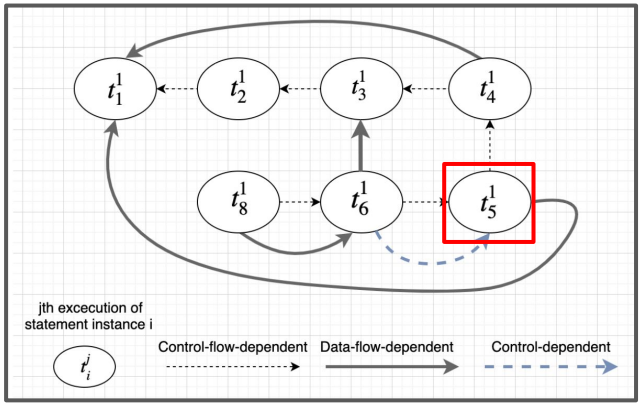
t' is empty (intra-method slicing)

Note: t' is the method containing t in inter-method slicing

$$t \leftarrow t_5^1,\ v \leftarrow m$$

$$W \leftarrow \{(t_3^1, p)\} \qquad slice \leftarrow \{t_8^1, t_6^1, t_5^1\}$$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6       p = p - 1;
7   }
8   assert(p == 4);
```



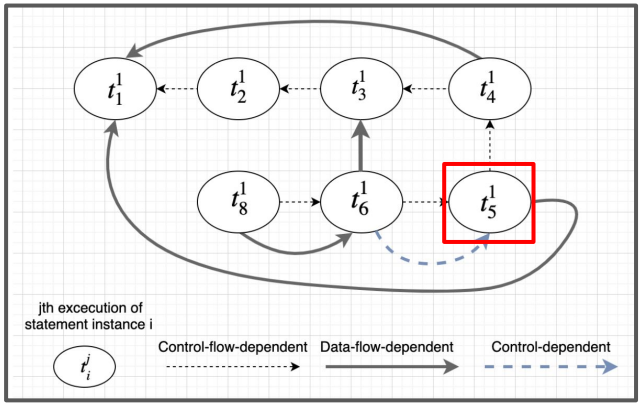
Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return sli
14 Procedure us    t)
15     return al   ariables used in t
```

$t \leftarrow t_5^1,\ v \leftarrow m$

$W \leftarrow \{(t_3^1, p), (t_1^1, m)\}$ $\quad slice \leftarrow \{t_8^1, t_6^1, t_5^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6       p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

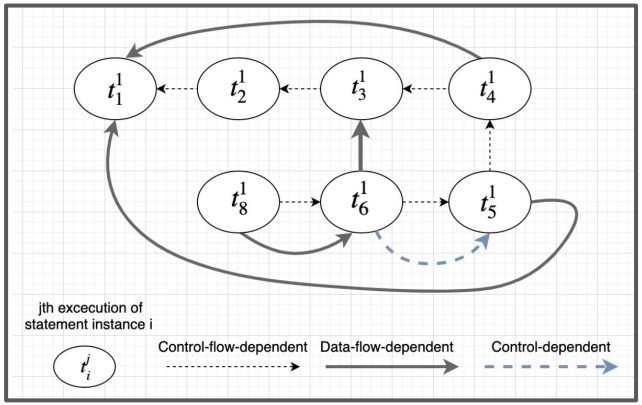
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure no processing statements more than once
9              W ← W ∪ {(t, v') | v' ∈ use(t')}
                   ▷ Add t to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t \leftarrow t_3^1,\ v \leftarrow p$

$W \leftarrow \{(t_1^1, m)\}$

$slice \leftarrow \{t_8^1, t_6^1, t_5^1, t_3^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

t' is empty (intra-method slicing)

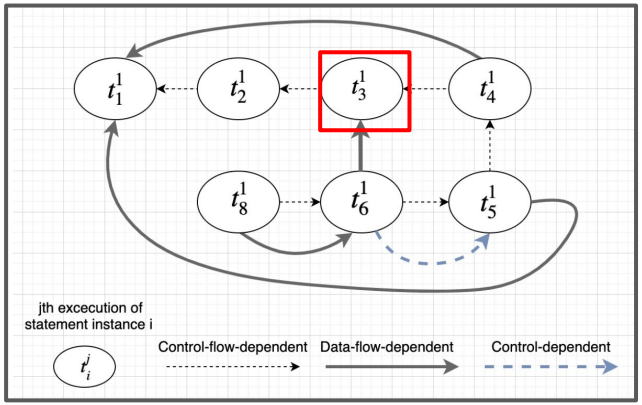
Note: t' is the method containing t in inter-method slicing

t' is empty in this case

$t \leftarrow t_3^1,\ v \leftarrow p$

$W \leftarrow \{(t_1^1, m)\}$ $\quad slice \leftarrow \{t_8^1, t_6^1, t_5^1, t_3^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

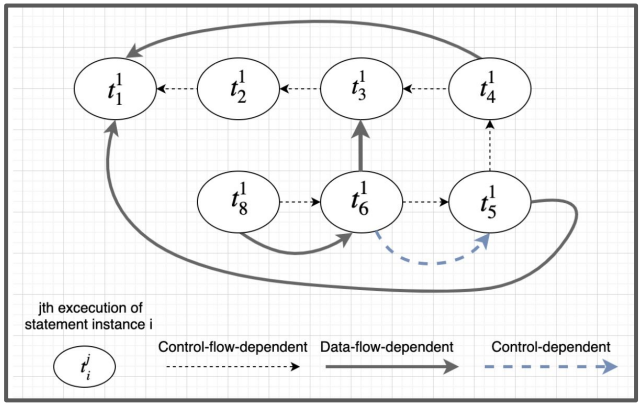
```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure no processing statements more than once
9              W ← W ∪ {(t, v') | v' ∈ use(t')}
                   ▷ Add t to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$t \leftarrow t_1^1,\ v \leftarrow m$

$W \leftarrow \{\}$

$slice \leftarrow \{t_8^1, t_6^1, t_5^1, t_3^1, t_1^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

t' is empty (intra-method slicing)

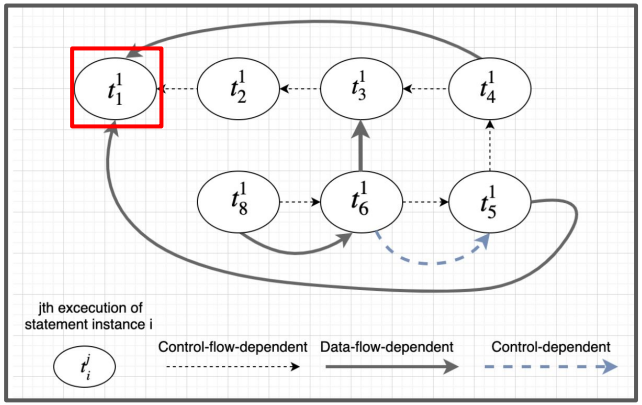
Note: t' is the method containing t in inter-method slicing

t' is empty in this case

$t \leftarrow t_1^1,\ v \leftarrow m$

$W \leftarrow \{\}$ $\quad slice \leftarrow \{t_8^1, t_6^1, t_5^1, t_3^1, t_1^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```

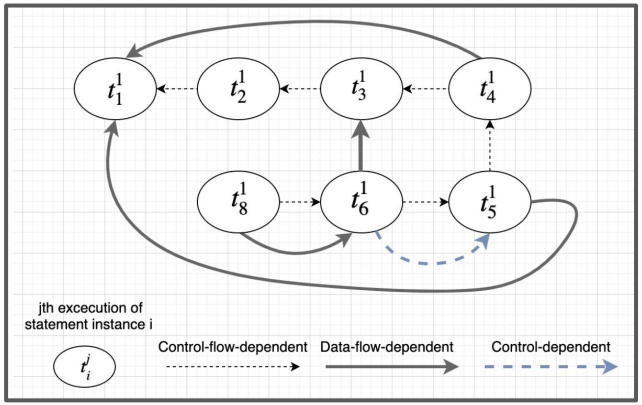


Algorithm 1: Dynamic slicing algorithm.

```
1  Input : DCFG, c, v1, ..., vn
   Output: slice
2  begin
       ▷ Create working set of pairs of statement instances and used
         variables
3      W ← {(c, v1) ... (c, vn)}
4      while W ≠ ∅ do
5          (t, v) ← pick and remove from W
6          slice ← slice ∪ {t}
7          t' ← the instance t is control-dependent on
8          if t' not in slice then
               ▷ Ensure not processing statements more than once
9              W ← W ∪ {(t', v') | v' ∈ use(t')}
                       ▷ Add t' to working set with all its used variables
10         t' ← reaching definition of v at t
11         if t' not in slice then
12             W ← W ∪ {(t', v') | v' ∈ use(t)}
13     return slice
14 Procedure use(t)
15     return all variables used in t
```

$W \leftarrow \{\}$ $\quad slice \leftarrow \{t_8^1, t_6^1, t_5^1, t_3^1, t_1^1\}$

```
1   int m = 1;
2   int n = 2;
3   int p = 4;
4   int q = m + 2;
5   if (m == 1) {
6         p = p - 1;
7   }
8   assert(p == 4);
```



References

H. Agrawal and J. R. Horgan, “Dynamic Program Slicing,” ACM SIGPLAN Notices, vol. 25, no. 6, pp. 246–256, 1990.

Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. IEEE International Conference on Software Testing, Verification and Validation (ICST), Distinguished Paper Award, 2021 (28% acceptance rate).

Khaled Ahmed, Mieszko Lis, and Julia Rubin. Slicer4J: A Dynamic Slicer for Java. ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), tools track, 2021.