A brief introduction to separation algorithm

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
static char* passwd = "sos";
int foo(){
  int x = 2, y = 3;
  x = x + y;
 printf("foo called\n");
 return 1;
}
int compare(char* str, char* passwd){
 char c = passwd[0];
  int ret = strcmp(str,passwd);
  int b = foo();
 if(ret == 0)
    return 1;
  else
    return 0;
}
int main(){
 char str[50];
  gets(str);
  int a = foo();
  if(compare(str, passwd) == 1)
    printf("correct passwd input!\n");
  else
    printf("wrong passwd input!\n");
  return 0;
}
```

Input:

PDG(*N*, *E*): a directed graph, where *N* is the set of nodes, *E* is the set of edges(data dependence edges, control dependence edges, call edges and edges for parameter interaction).

Func_list: a list which stores all functions in the input module

start_nodes: a given set of privileged sensitive data nodes(in demo programs, usually represented as global variables for simplicity)

Queue :an queue as the working list for the expansion computation

colored_nodes: a set that includes all colored nodes.

Pseudocode

```
(start_nodes initialization)
begin:
     colored_nodes = start_nodes;
     working node = null;
     for each node n in start nodes
          Push(Queue, n);
          colorn and insert n into colored nodes;
     while (!IsEmpty(Queue)) do
          working_node = Pop(Queue);
          for each successor node sn of the working node
             if sn uncolored && [n->sn] is NOT a control dependence edge
                     color sn and insert sn into colored nodes.
                     Push(Queue, sn);
          end for
     end while
     for each element in colored nodes
         color corresponding functions in Func_list;
end
```

Output: two function sets, one includes all colored function, the other includes uncolored ones.

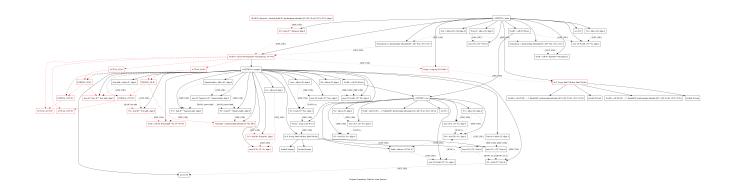


Figure 1. A PDG-based seperation for the sample program in page 1.

We start from the sensitive node @passwd =... and use BFS coloring algorithm to color all its successor nodes that can be reached through non-control dependence edges. In this demo, the two function sets we get in the end are {main, compare} and {foo}.