Towards an automated proof of the CNDFS algorithm

Stéphane Aubry

Franck Butelle, Sami Evangelista, Micaela Mayero

LIPN - USPN (Paris 13) - France

GDR Scalp&Vérif, 19/11/2024, Lille

Motivations

- Formal proof of a critical verification algorithm
- Goal: CNDFS algorithm (Concurrent Nested Depth-First Search)
- First step: NDFS algorithm (Nested Depth-First Search)
- State of the art:
  - a parallel NDFS algorithm (Oortwijn & al, [TACAS2020]) proved in Vercors
  - Formal Proofs of Tarjan's Algorithm in Why3, Coq, and Isabelle (Chen & al)
- Interaction between verification, concurrency and formal proof research areas

LTL model checking using the NDFS algorithm

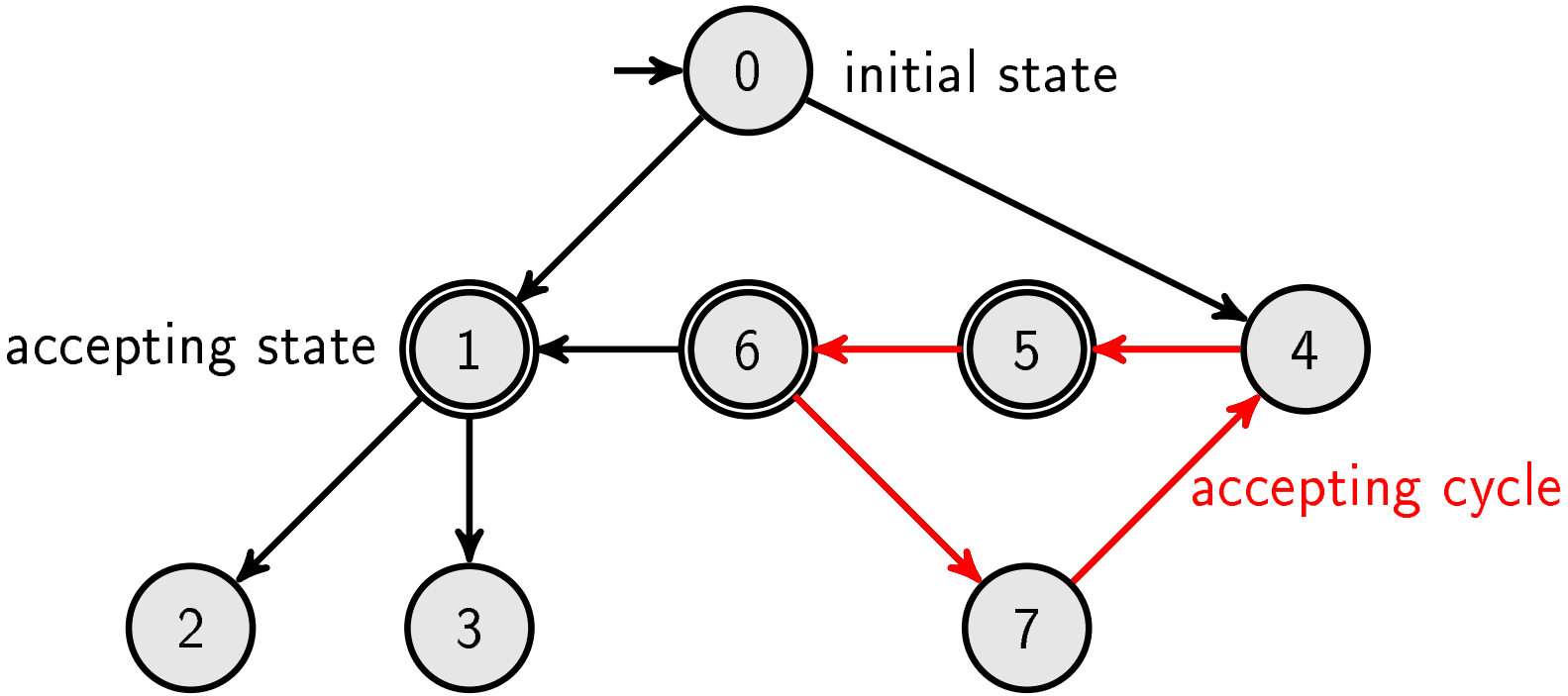
Formal proofs

- Tool choice
- Vercors
- F*

CNDFS

Conclusion and future work

The LTL model checking problem

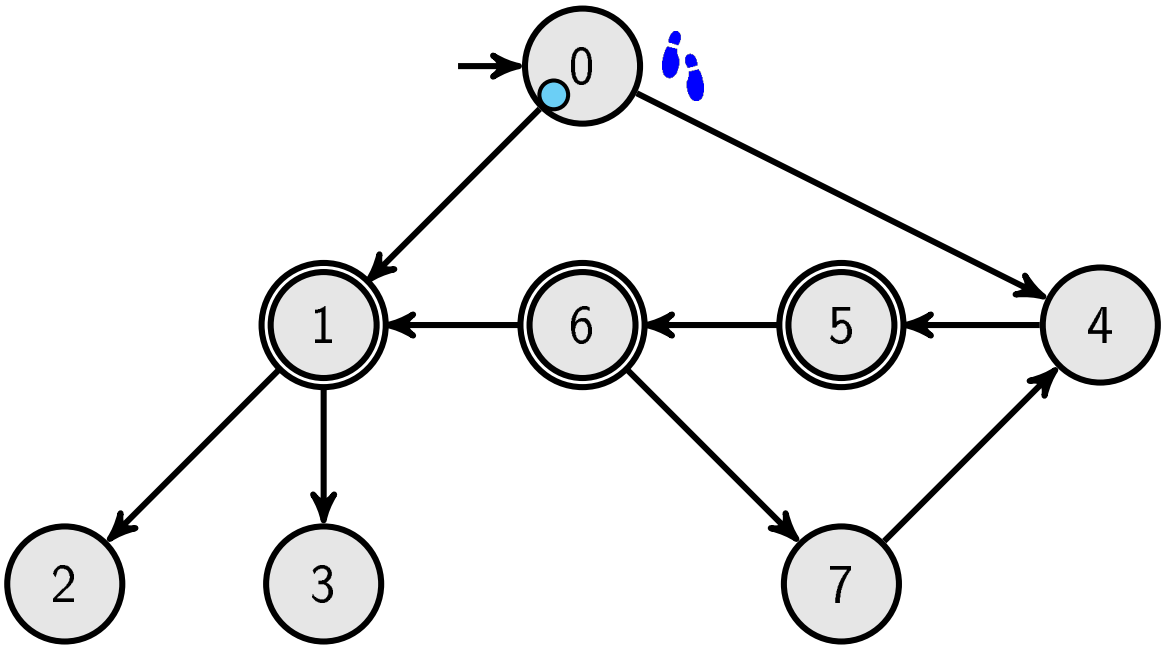


- given a directed graph
- having accepting states
- find an accepting cycle (i.e., looping on an accepting state)
- accepting cycle $\Leftrightarrow$ bad behaviour of the system

NDFS algorithm — Principle

- perform nested depth-first searches:
  - a blue dfs looks for accepting states
  - before backtracking from an accepting state: initiate a red DFS
  - the red DFS looks for cycles around accepting states
- mark states to detect cycles and remember visited states:
  - cyan $\Leftrightarrow$ state on the search stack of the blue DFS
  - blue $\Leftrightarrow$ visited and backtracked state from the blue DFS
  - red $\Leftrightarrow$ visited state from the red DFS
- an accepting cycle is found when the red DFS reaches a cyan state

NDFS algorithm — Execution example

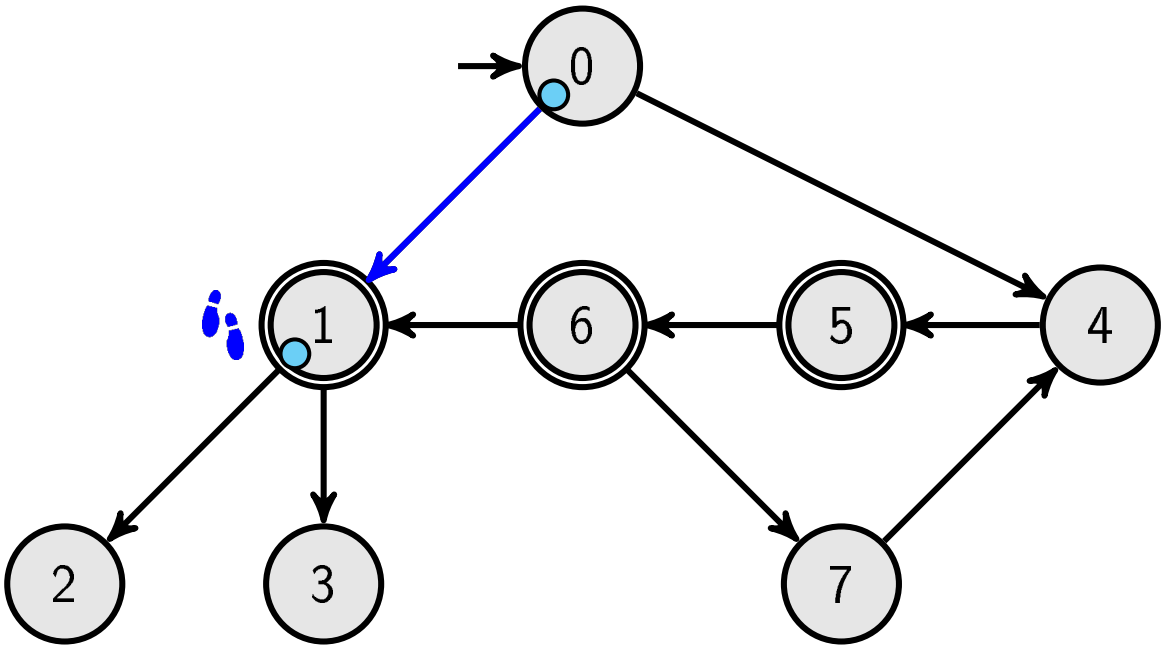


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

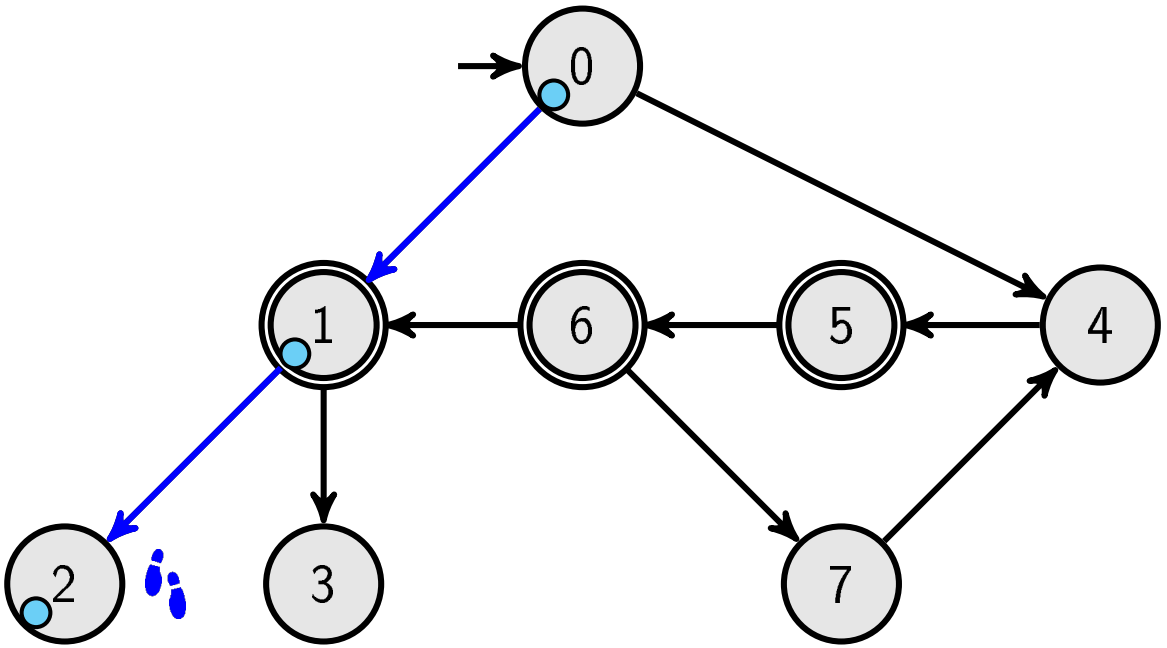


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

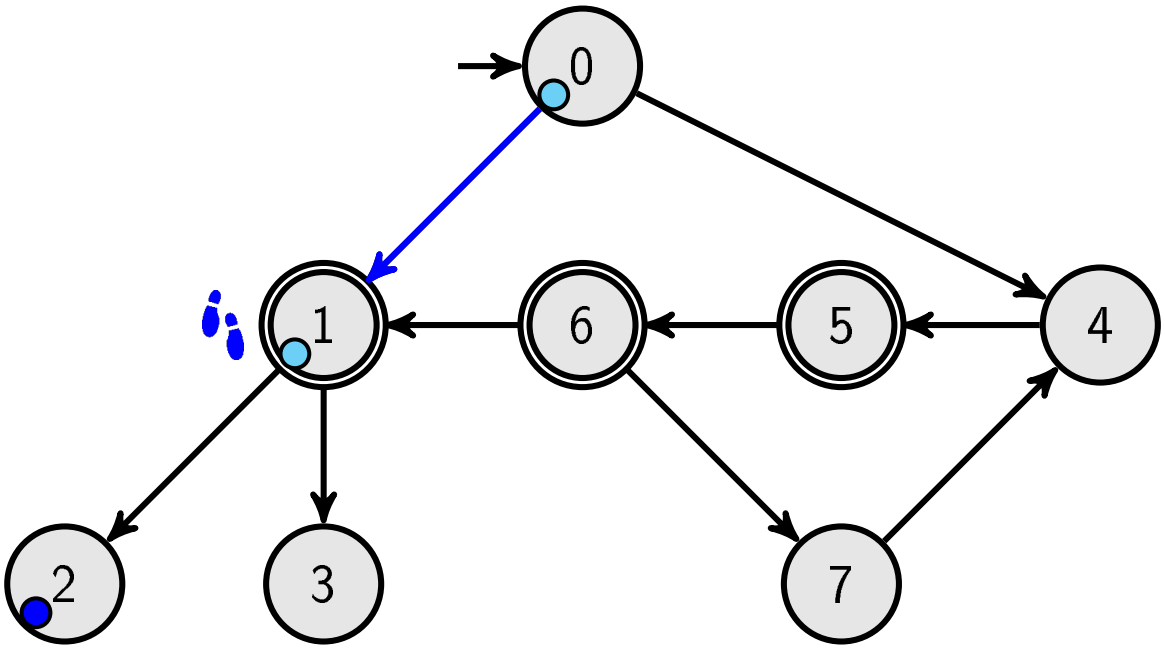


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

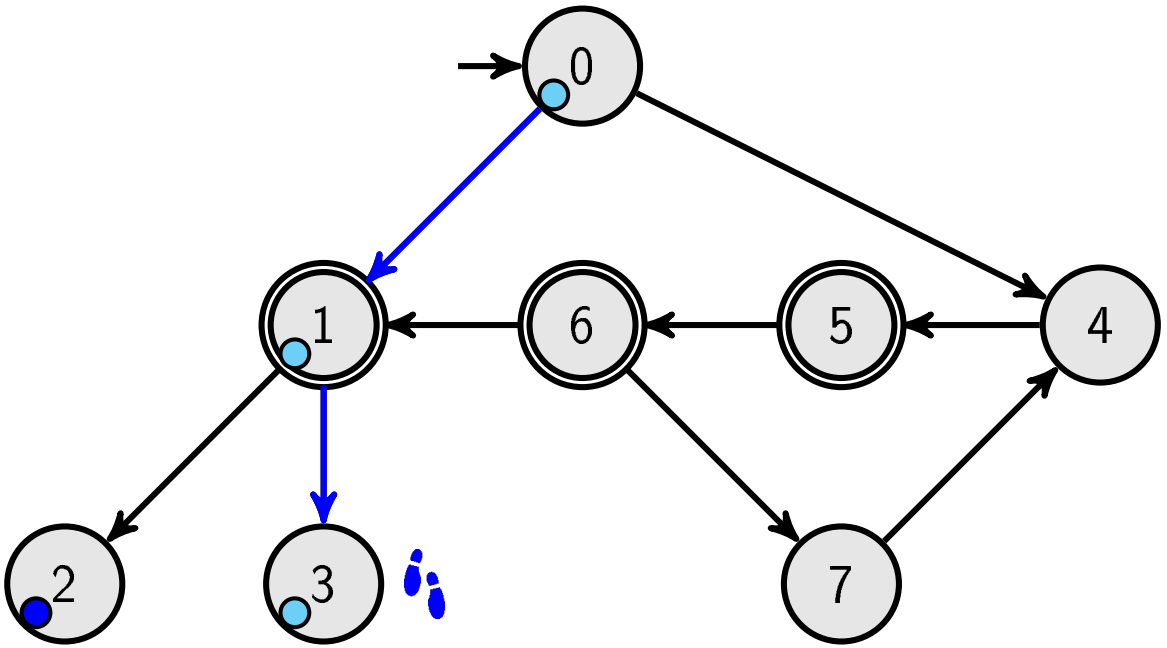


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

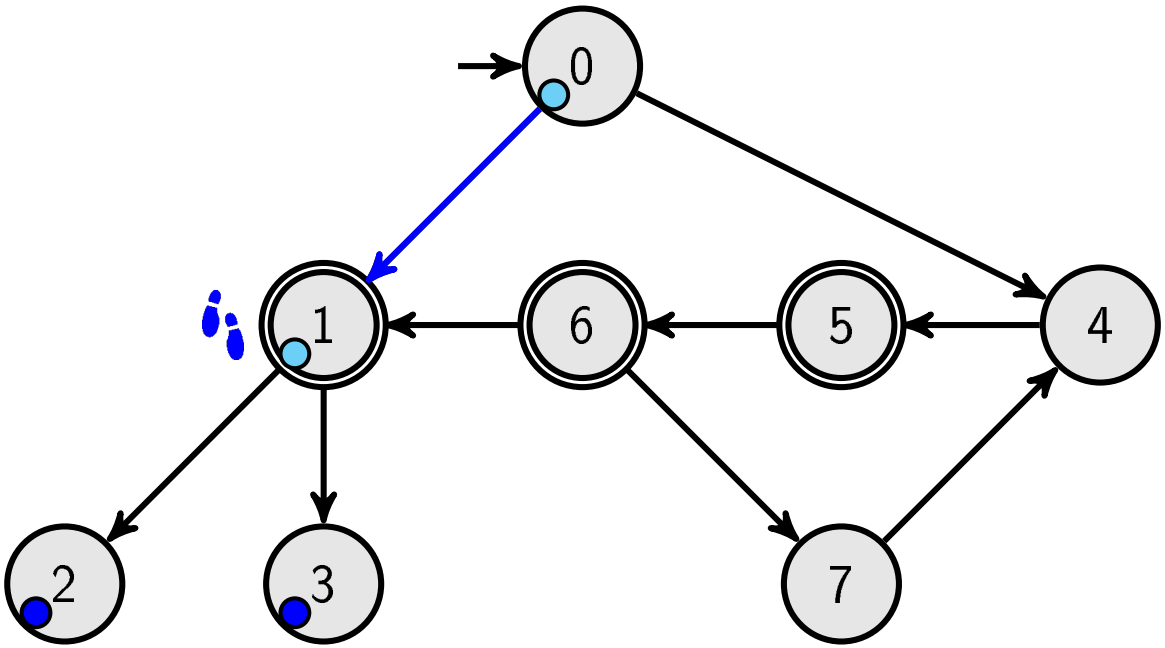


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

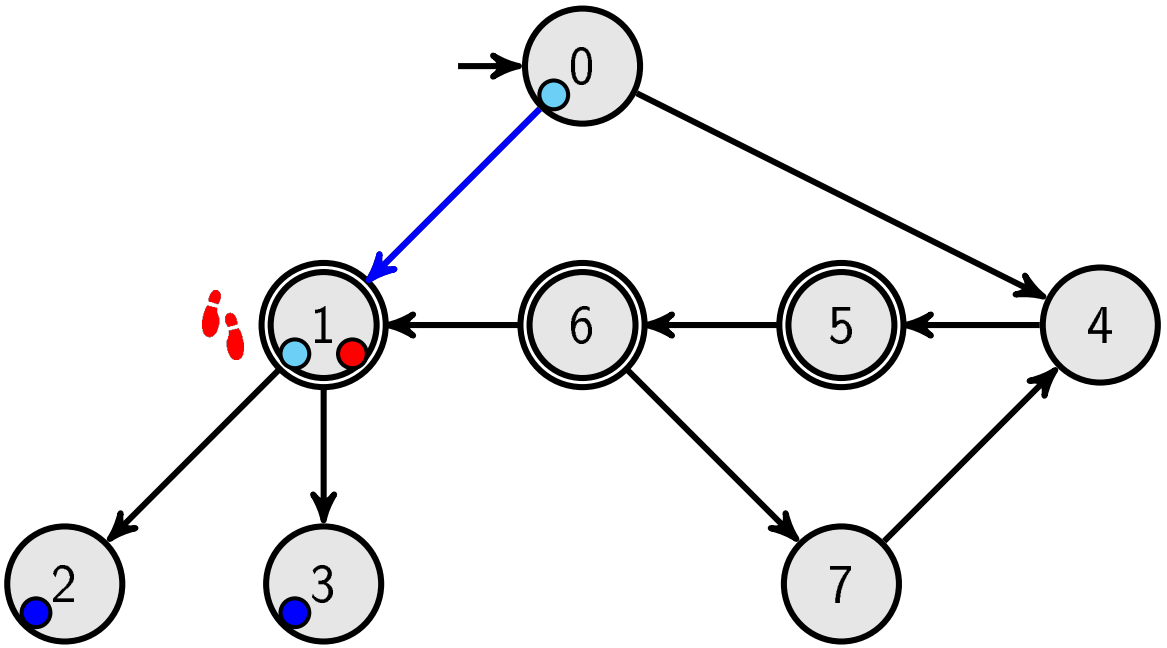


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

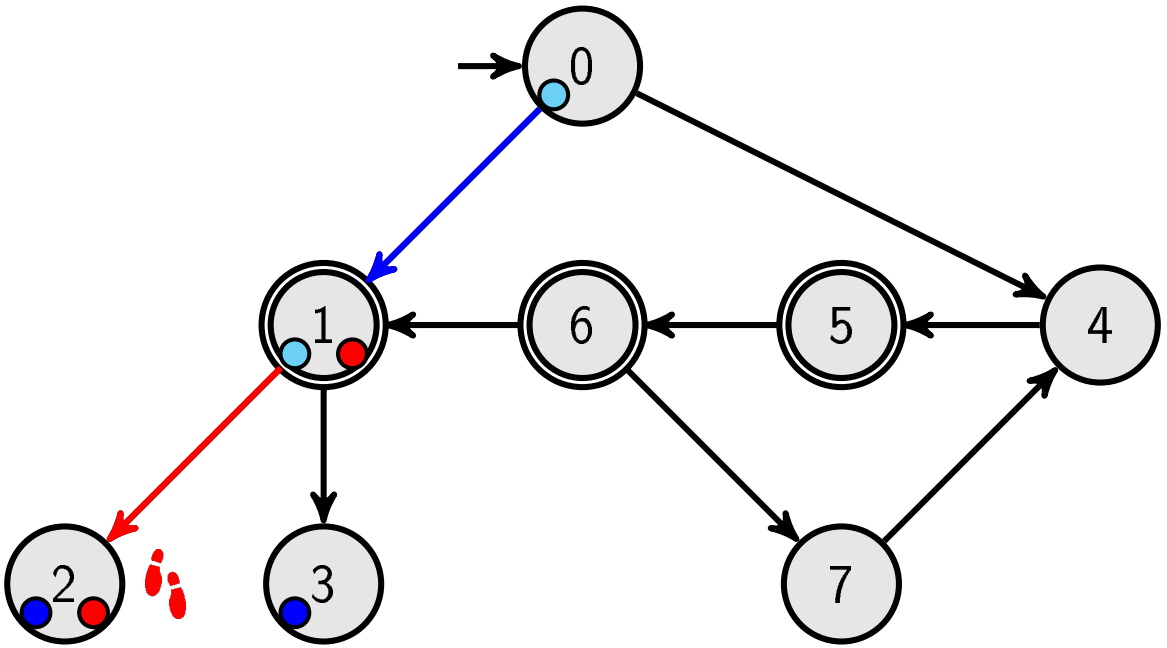


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

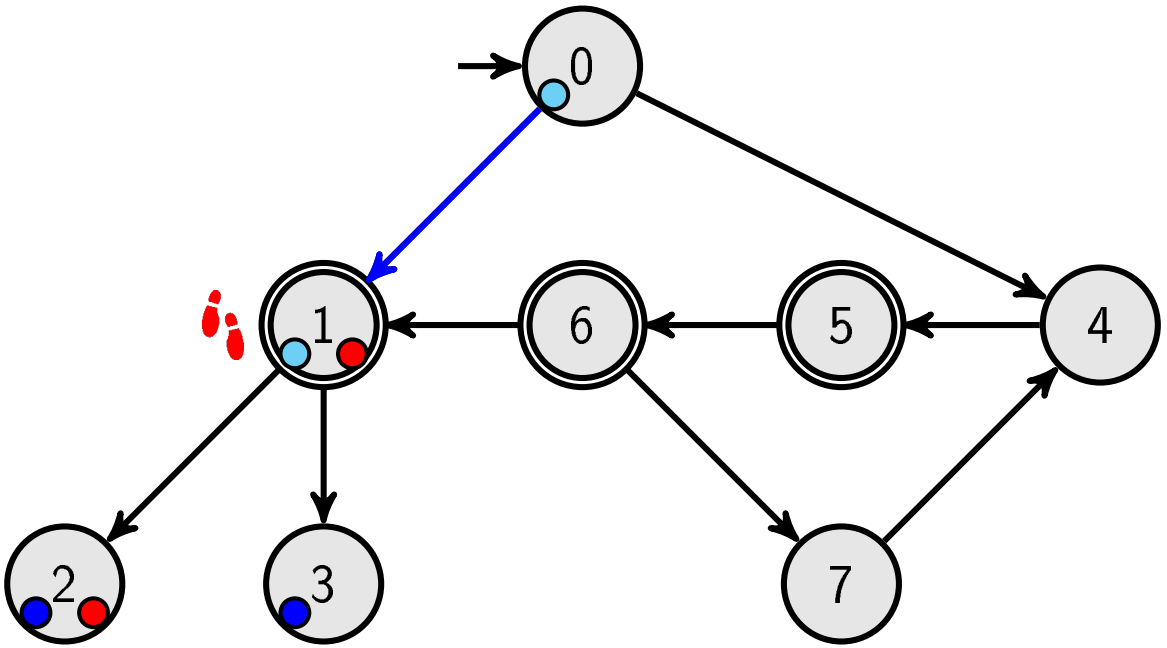


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

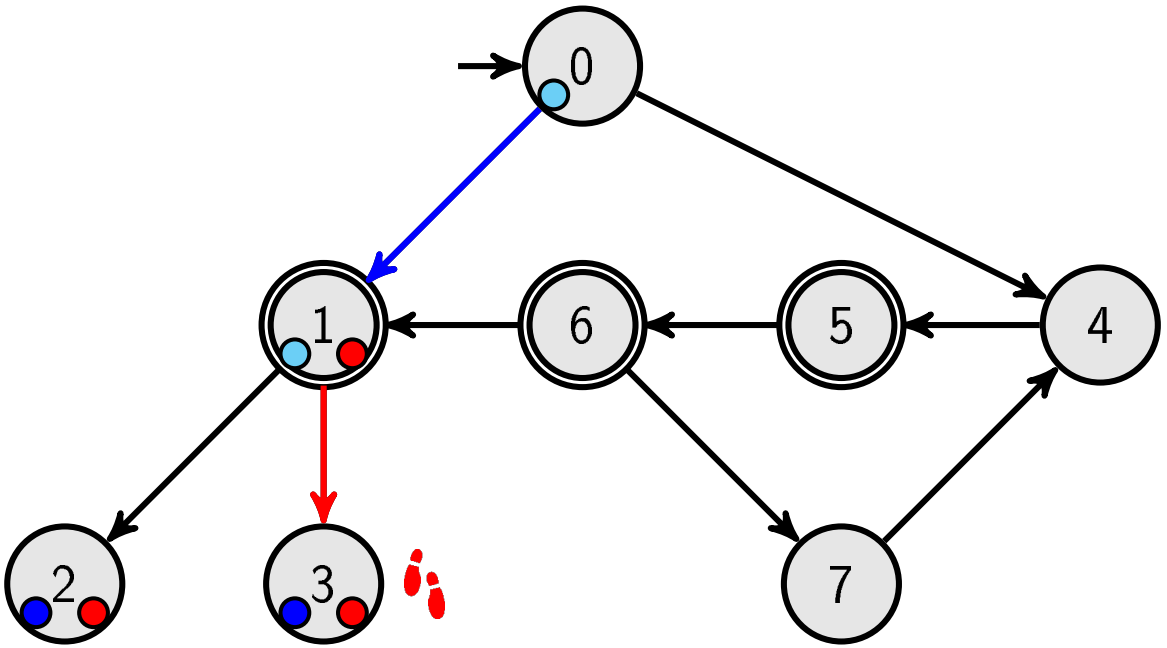


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

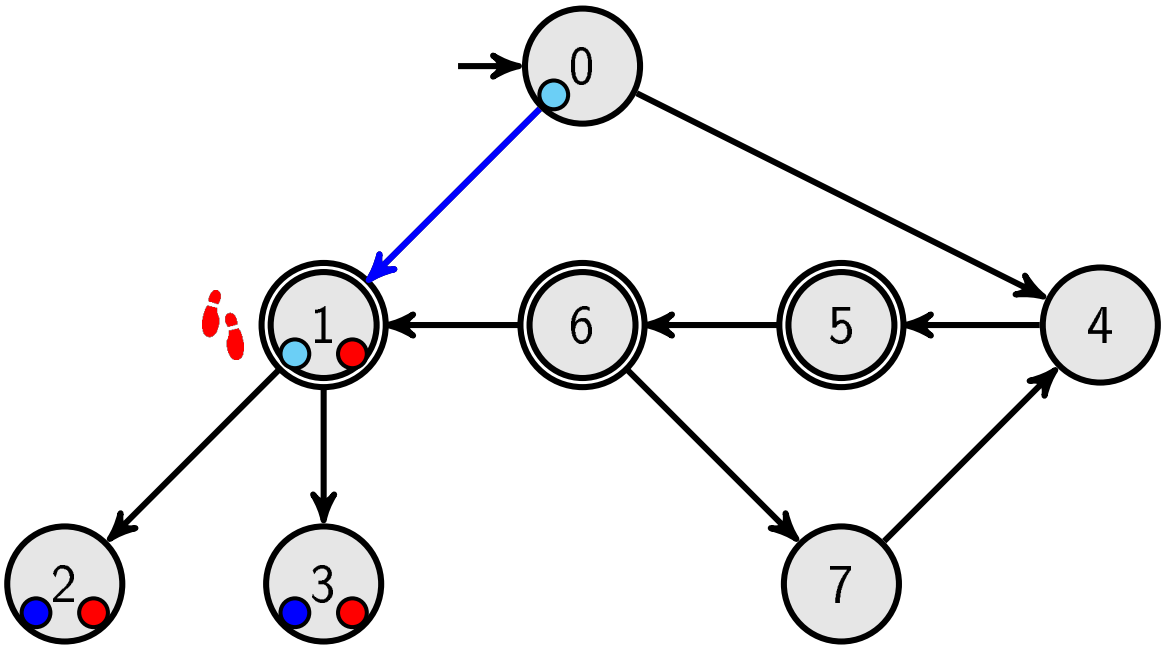


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

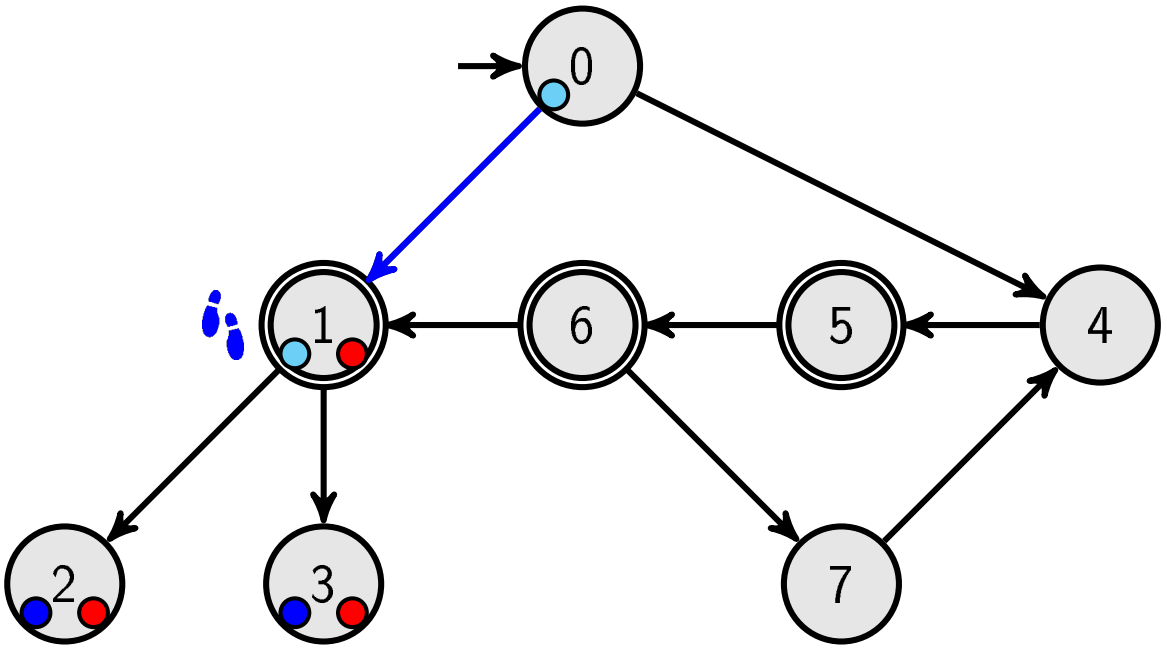


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

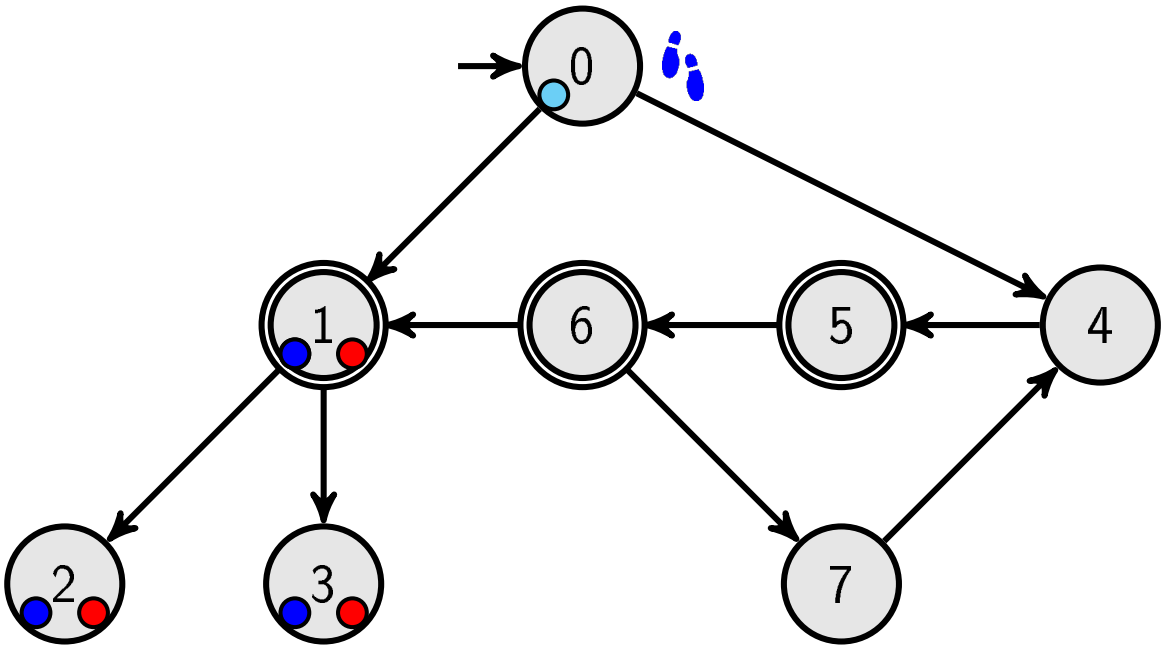


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

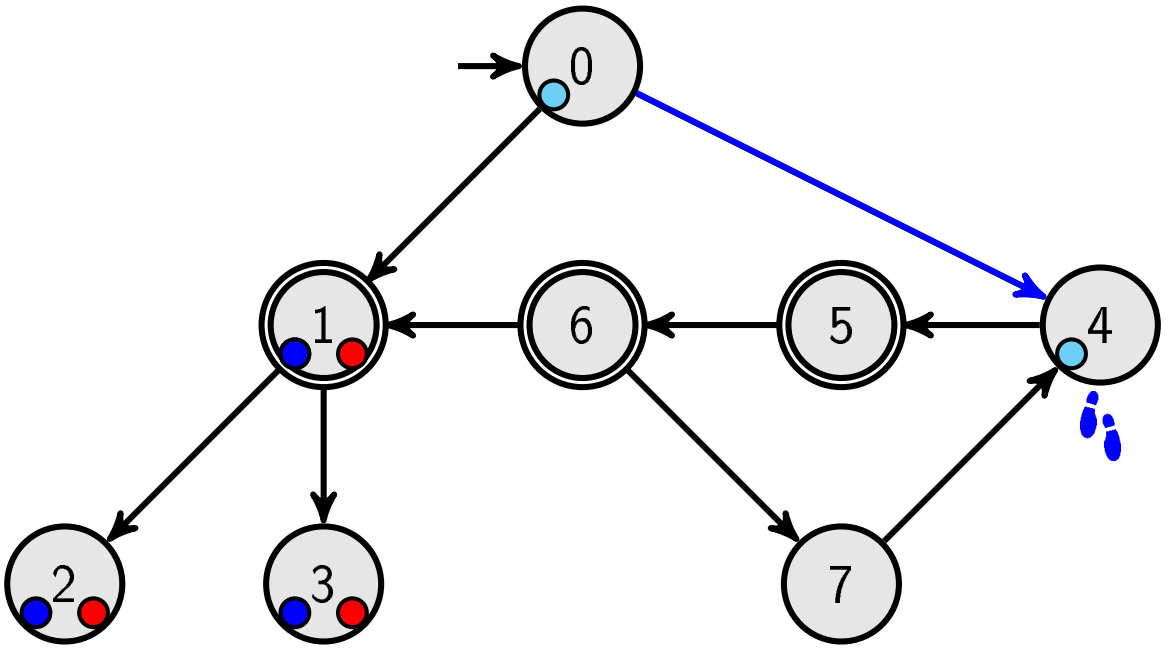


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

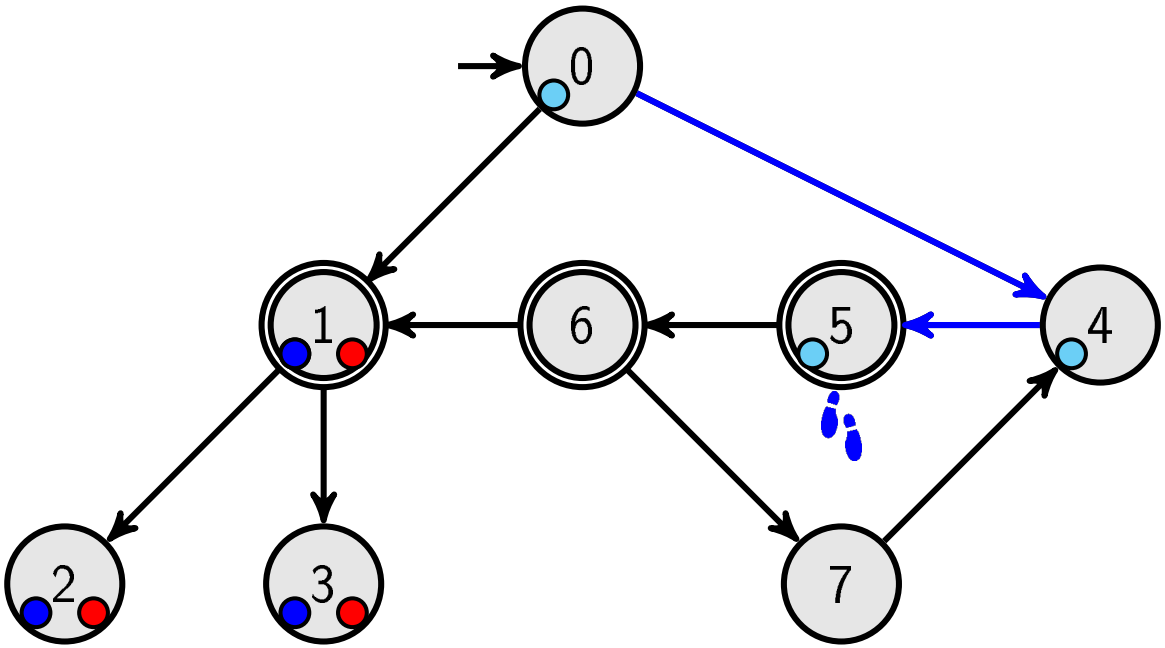


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

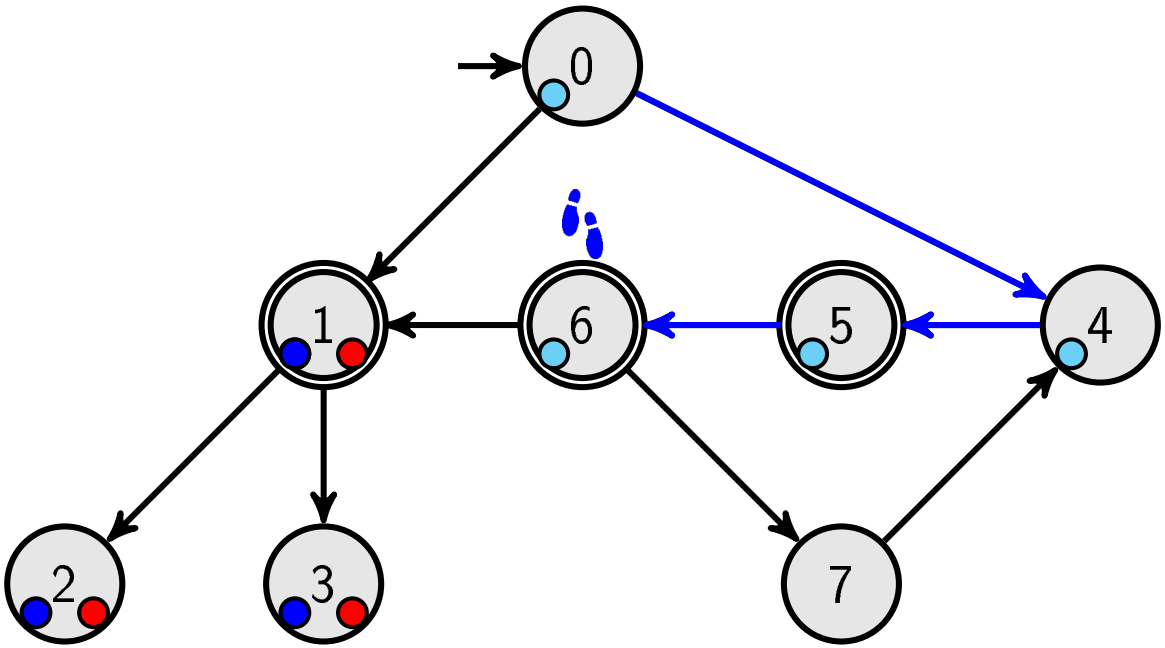


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

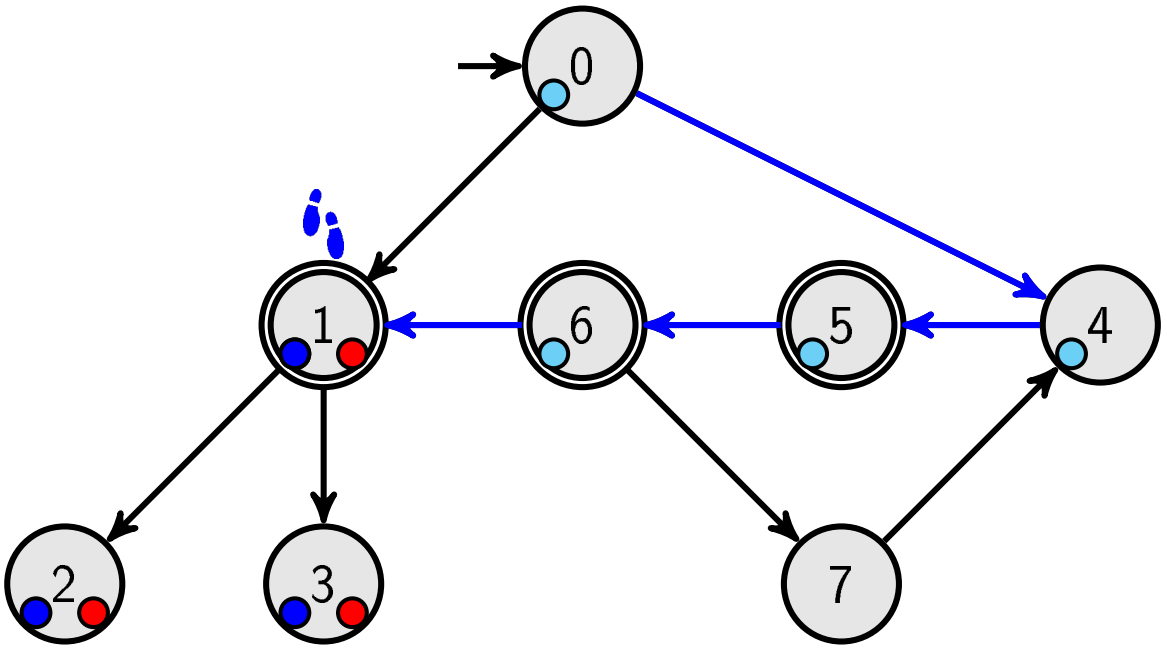


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

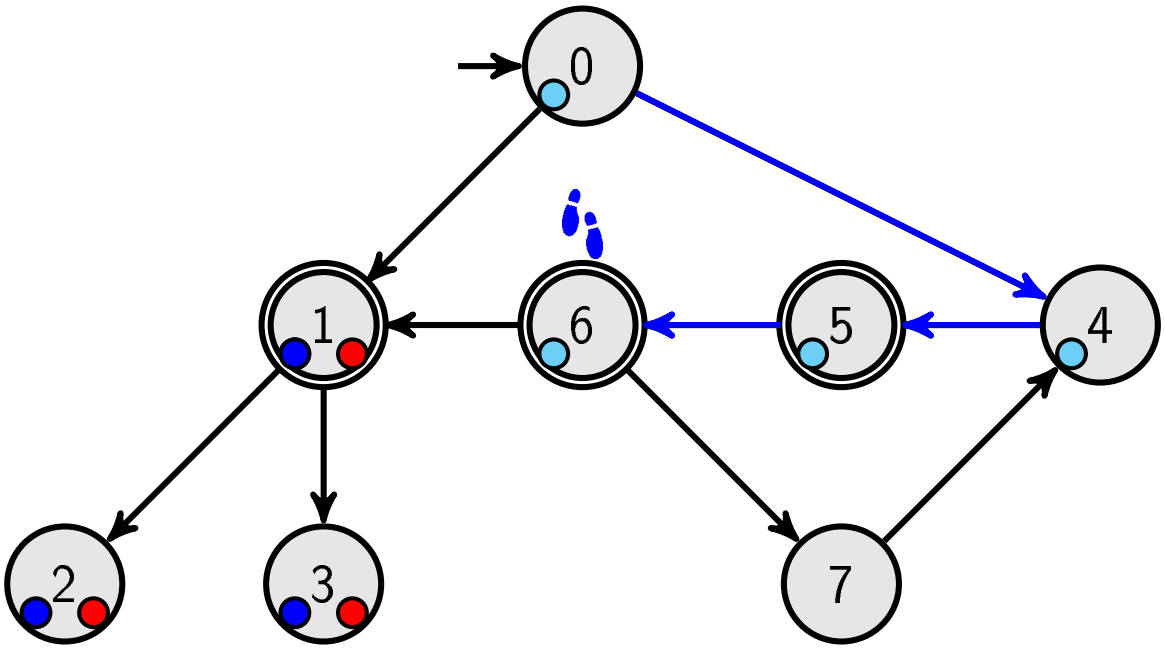


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

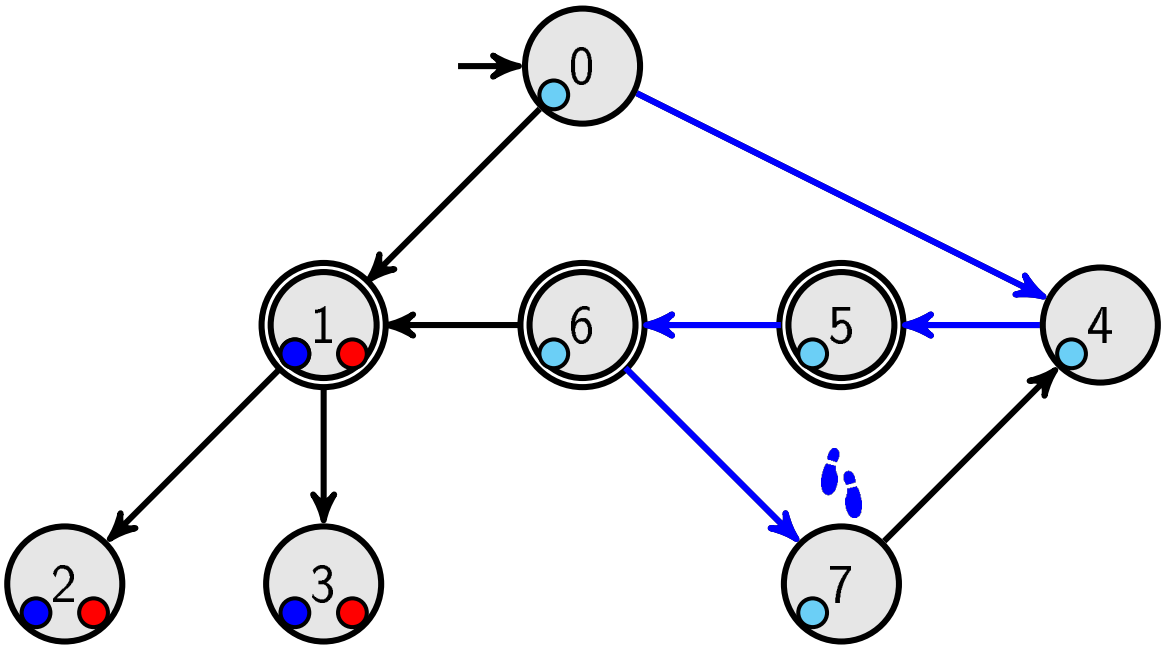


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

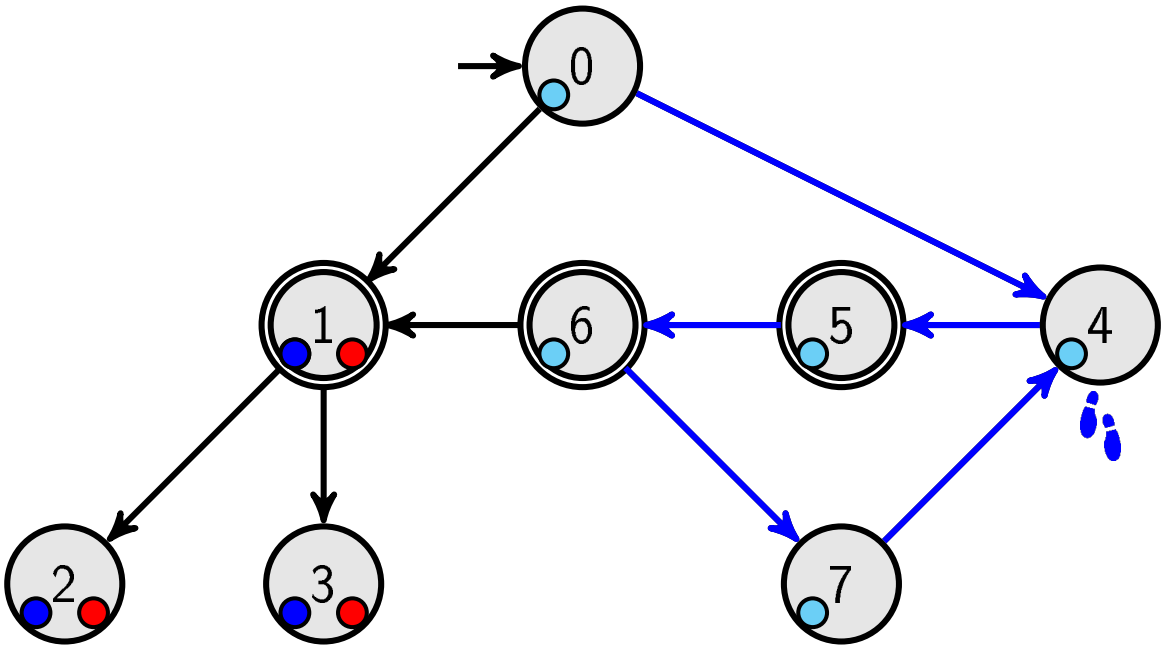


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

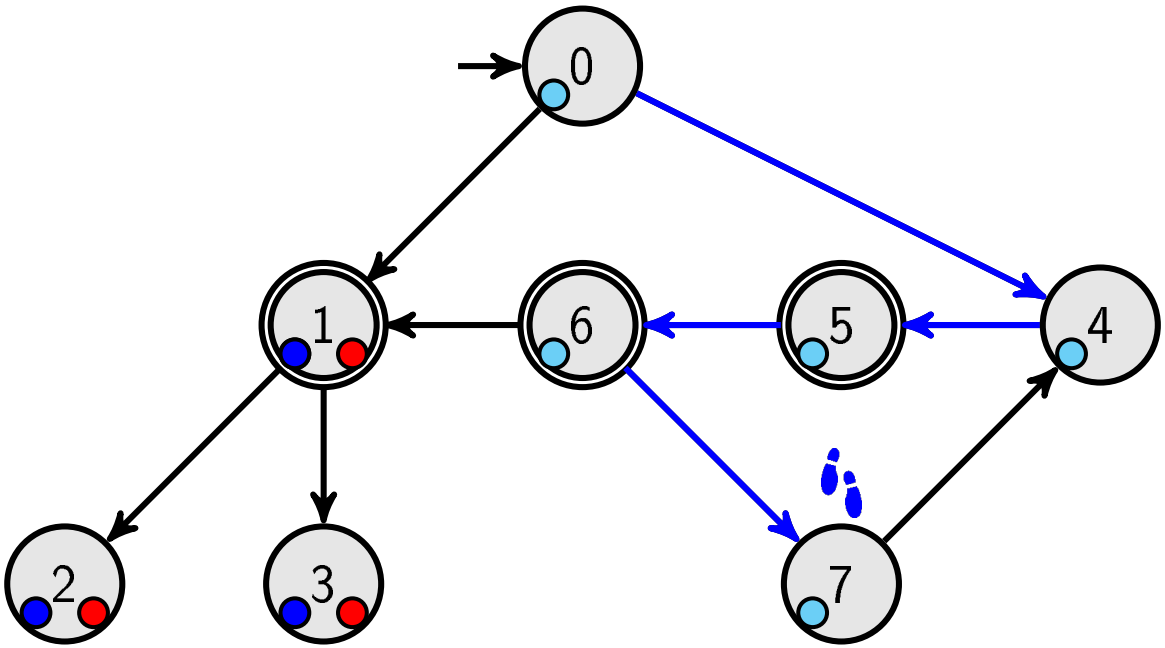


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

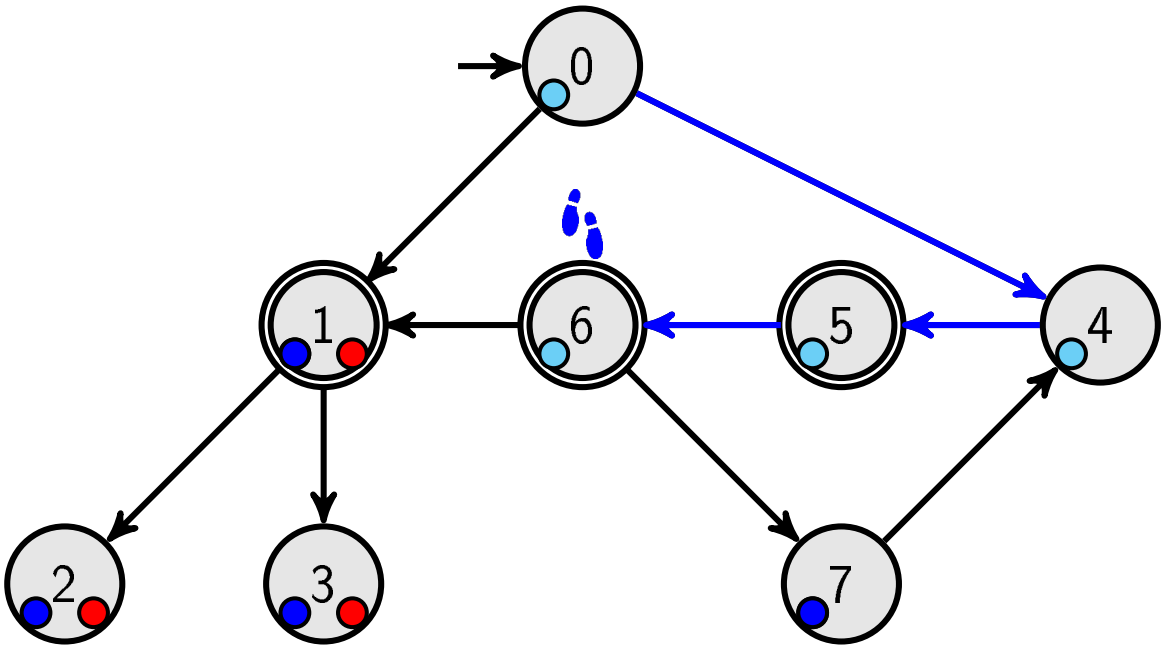


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

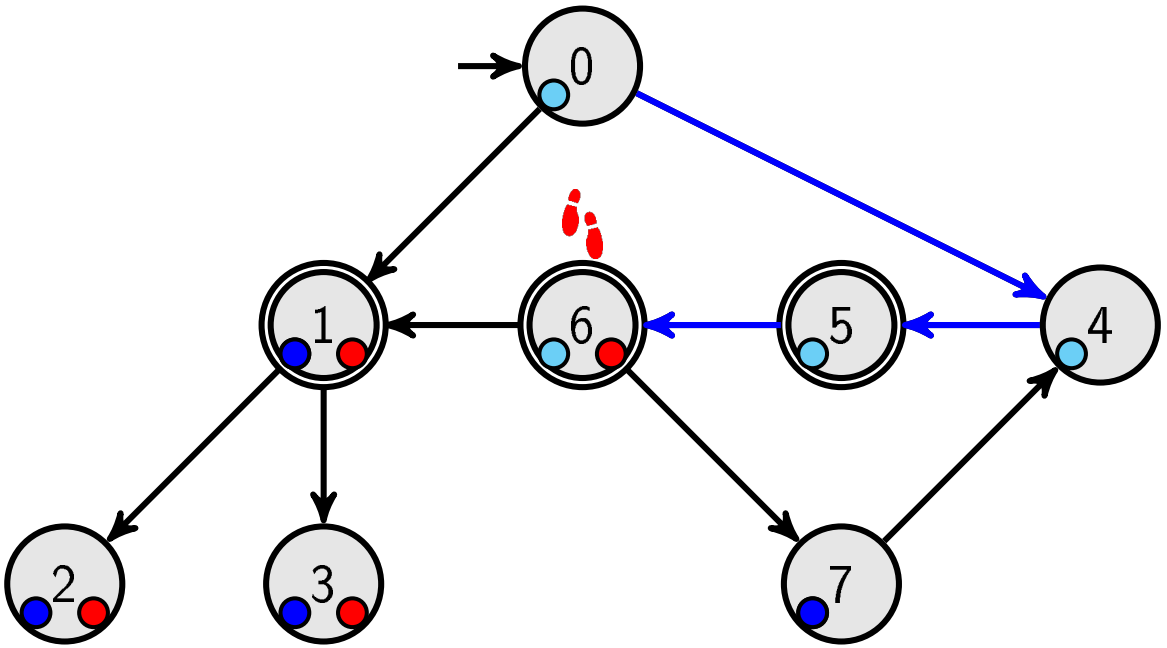


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

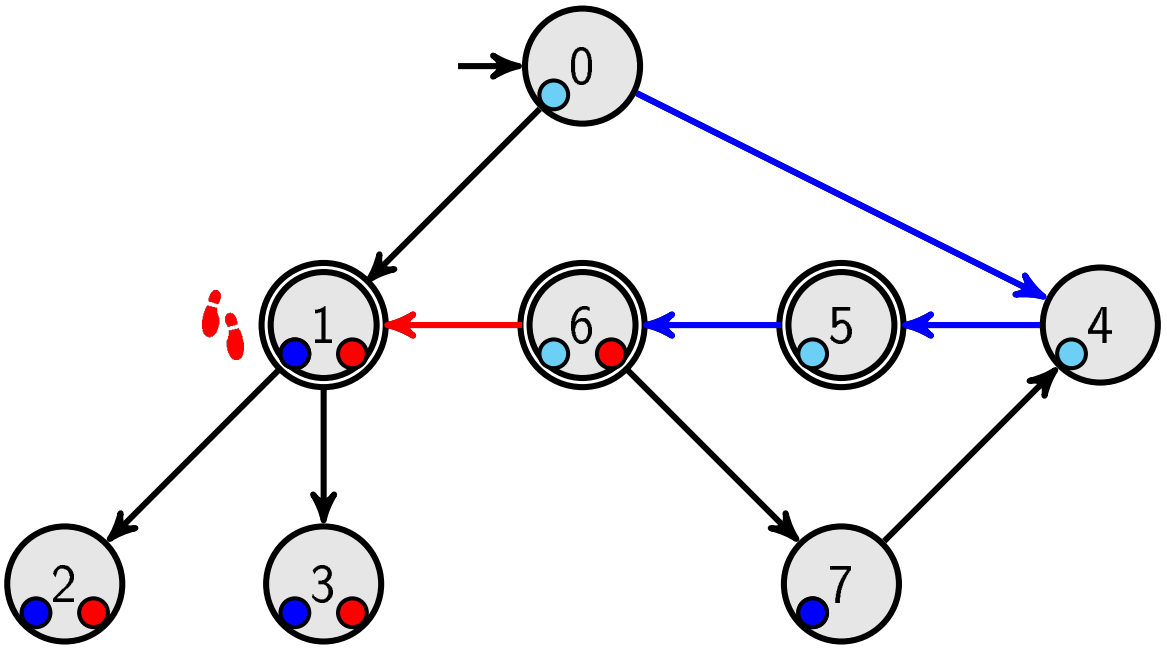


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

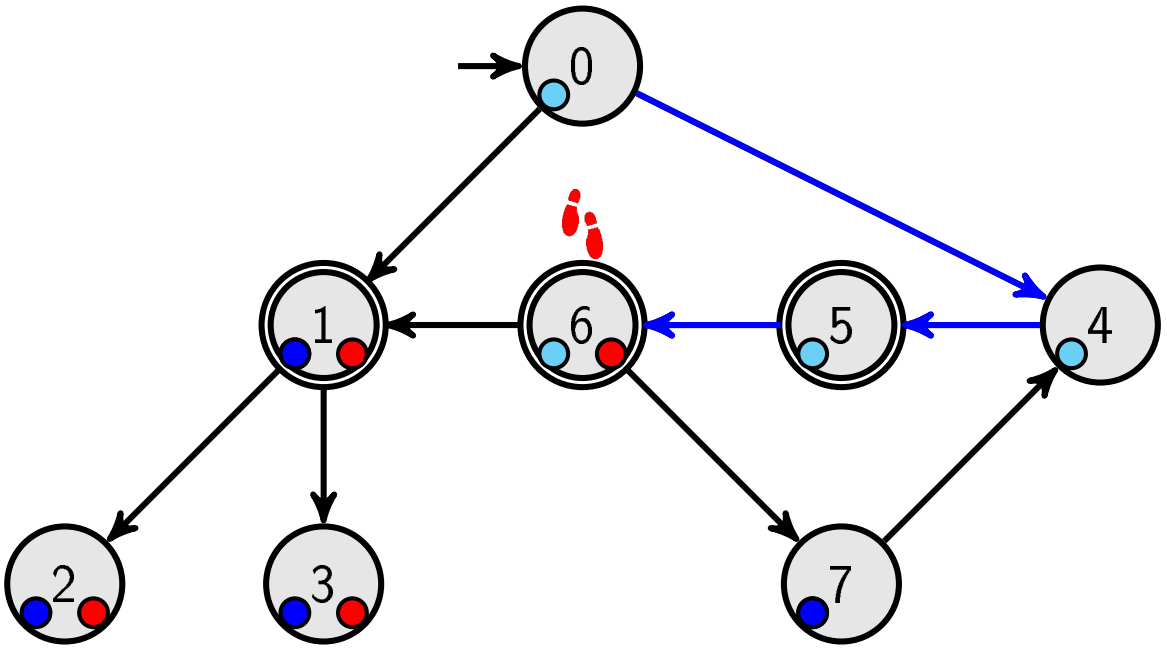


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

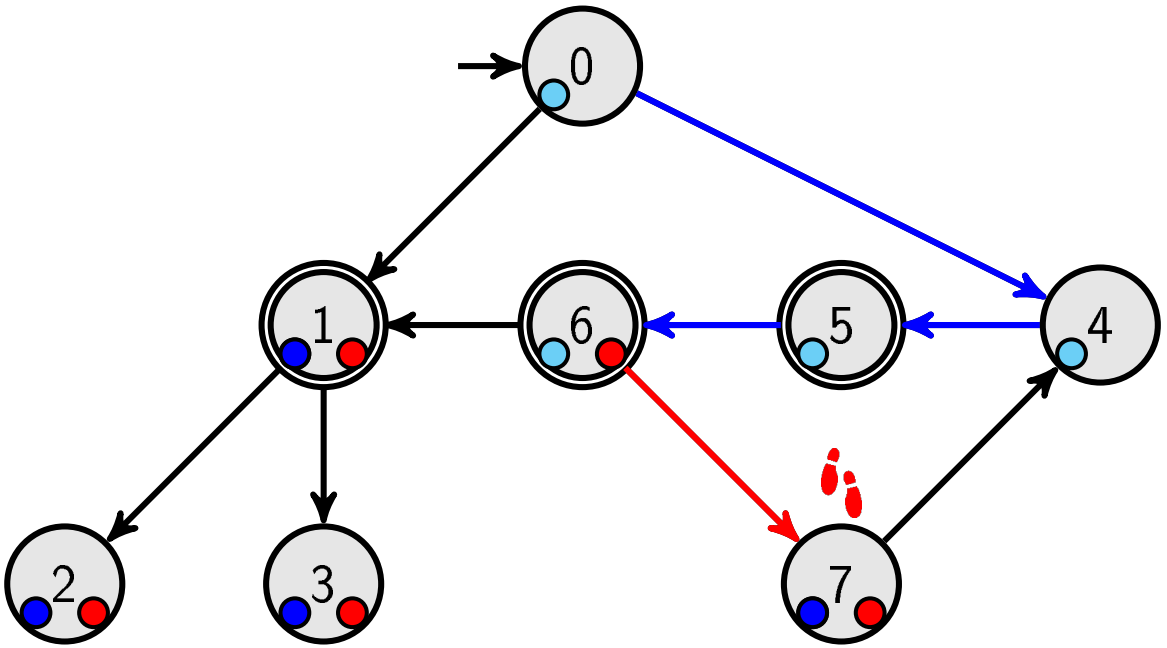


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example

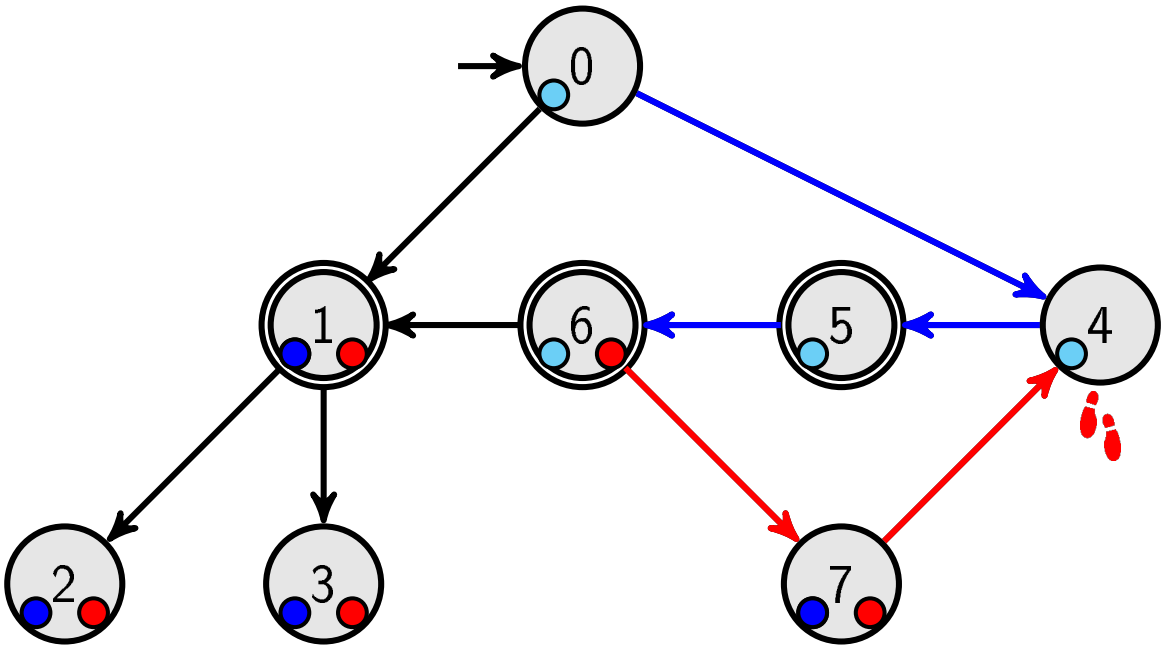


cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

NDFS algorithm — Execution example



cyan ⇔ on the blue stack

blue ⇔ left the blue stack

red ⇔ entered the red stack

Red DFS reaches a cyan state ⇒ accepting cycle detected !!!

Table of contents

Tool choice

COQ

Pros	Cons
proof assistant possible extraction	not suitable for concurrency termination of functions

WHY3

Pros	Cons
easy to write program	not suitable for concurrency

VERCORS

Pros	Cons
suitable for concurrency cf state of the art	significant overhead (permissions. . .) some too generic error messages

F* (in progress)

Pros	Cons
suitable for concurrency pulse ?	??

Table of contents

Presentation of VerCors

- verification tool that uses static analysis
- it allows us to prove partial correctness of a given piece of code
- it requires users to add annotations in the code (which specifies the expected behaviour)
- it uses the permission-based Concurrent Separation Logic (CSL)
- it supports Java, C (incl. OpenMP), OpenCL and PVL

VerCors basic example: initializing array with zeros in PVL

```
1  context_everywhere A != null;
2  context_everywhere
3     (\forall * int j; 0 <= j && j < A.length; Perm(A[j], write));
4  ensures (\forall int j; 0 <= j && j < A.length; A[j] == 0);
5  void clear(int[] A) {
6       int i = 0;
7
8     loop_invariant 0 <= i && i <= A.length;
9     loop_invariant (\forall int j; 0 <= j && j < i; A[j] == 0);
10
11      while (i < A.length) {
12           A[i] = 0;
13           i = i + 1;
14      }
15 }
```

Example: dfs-red (1/4) algorithm

```
procedure dfsRed(s) is
    s.red := true
    for all s' in succ(s) do
        if s'.cyan then exit(cycle)
        if not s'.red then dfsRed(s')
```

Example: dfs-red in PVL (2/4) without annotation

```
1  option<int> dfs_red (BuchiGraph g, Colors c, int s0, int s) {
2      c.add_red(s);
3      seq<int> _succ = g._IM()[s];
4      int  sp = 0, i = 0;
5      option<int> res = None;
6      while(res == None && i < |g._IM()[s]|) {
7        sp = g._IM()[s][ i];
8        if (sp in  c._cyan()) {res = Some(s0);}
9        else  if  (!( sp in  c._red())){res = dfs_red(g, c, s0, sp);}
10       if (res == None) {i = i + 1;}
11     }
12     return res;
13   }
```

Example: dfs-red in PVL (3/4) with annotations

```
class DfsRed {

  constructor () {}

  context_everywhere g.state();
  context_everywhere c.state();
  context_everywhere g.is_valid();
  context_everywhere g.is_valid_state(s);
  context_everywhere g.is_valid_state(s0);
  context_everywhere g.is_valid_colors(c);
  // la racine de la dfs_red (i.e., l'état d'où la dfs_red a été initiée est acceptant)
  context_everywhere s0 in g._A();
  // tous les états cyans mènent à la racine de la dfs_red
  requires (\forall int x; g.is_valid_state(x) && x in c._cyan(); x == s0 || g.is_reachable(x, s0));
  ensures \old(g._A()) == g._A();
  ensures \old(g._IM()) == g._IM();
  // s est rouge apres l'appel
  ensures s in c._red();
  // l'ensemble red ne peut que croitre
  ensures \old(c._red()) <= c._red();
  // dfs_red ne change pas l'ensemble des etats bleus ni cyans
  ensures \old(c._blue()) == c._blue();
  ensures \old(c._cyan()) == c._cyan();
  // si le resultat est None (pas de cycle acceptant accessible depuis s)
  //    => tout successeur de s est rouge
  ensures (\result == None) ==>
    (\forall int t; t in g._IM()[s]; t in c._red());
  requires (s == s0) || (g.is_reachable(s0, s));
  //lemmes finaux :
  // si on renvoie Some _ alors on a un cycle autour de s0
  ensures (\result != None) ==> g.acc_cycle_from(s0);
  //ensures (\result == None) ==>
  //  (\forall int sp; g.is_valid_state(sp); g.is_reachable(s, sp) ==> sp in c._red());
  option<int> dfs_red
    (BuchiGraph g,
     Colors c,
     int s0,
     int s) {
    c.add_red(s);
    seq<int> _succ = g._IM()[s];
    int sp = 0, i = 0;
    option<int> res = None;
    loop_invariant \old(g._IM()) == g._IM();
    loop_invariant \old(g._A()) == g._A();
    loop_invariant \old(s) == s;
    loop_invariant \old(s0) == s0;
    loop_invariant i >= 0 && i <= |g._IM()[s]|;
    loop_invariant s in c._red();
    loop_invariant \old(c._red()) <= c._red();
    loop_invariant \old(c._blue()) == c._blue();
    loop_invariant \old(c._cyan()) == c._cyan();
    //loop_invariant (\forall int j; 0 <= j && j < i;
    //   (\forall int sp; g.is_valid_state(sp); g.is_reachable(g._IM()[s][j], sp) ==> sp in c._red()));
    loop_invariant (s == s0) || g.is_reachable(s0, s);
    loop_invariant g.is_valid_state(sp);
    //loop_invariant g.is_reachable(s0, sp);
    //loop_invariant (s == s0) || is_reachable(s, sp);
    //loop_invariant (s == s0) || is_reachable_trans(s0, s, sp);
    loop_invariant (res != None) ==> g.acc_cycle_from(s0);
    loop_invariant (res == None) ==> (\forall int j; 0 <= j && j < i; g._IM()[s][j] in c._red());
    while(res == None && i < |g._IM()[s]|) {
      sp = g._IM()[s][i];
      assert res == None;
      assert g.is_valid_state(sp);
      assert sp in g._IM()[s];
      assert g.path(seq<int> {s, sp}, s, sp);
      assert g.lemma_is_reachable(s, sp);
      assert g.is_reachable(s, sp);
      assert (s == s0 || g.is_reachable(s0, s));
      assert g.lemma_reachability_is_transitive();
      assert g.is_reachable(s0, sp);
      if(sp in c._cyan()) {
        assert g.is_reachable(s0, sp) && (sp == s0 || g.is_reachable(sp, s0));
        assert g.is_reachable(s0, s0);
        assert g.acc_cycle_from(s0);
        res = Some(s0);
      }
      else if(sp in c._red()) {
      } else {
        res = dfs_red(g, c, s0, sp);
      }
      assert (sp in c._red()) || (sp in c._cyan());
      if(res == None) {
        i = i + 1;
      }
    }
    return res;
}
```

Example: dfs-red in PVL (4/4); zoom

```
1   // if the result is None
2   // (no accepting cycle reachable from s)
3   // => any successor of s is red
4
5  ensures (\result == None) ==>
6      (\forall int t; t in g._IM()[s]; t in c._red());
7  requires (s == s0) || (g.is_reachable(s0, s));
8
9   // final lemmae :
10  // if the result is Some _ then there is a cycle around s0
11
12 ensures (\result != None) ==> g.acc_cycle_from(s0);
```

VerCors proof status

- ☑ blue states have blue or cyan successors
- ☑ at the end of NDFS algo[1], there is no more cyan state
- ☑ if dfs_red finds a cycle around a state then it is an accepting cycle
- ☐ at the end of NDFS algo[1], every blue state has blue successor
- ☐ red states have red successors at the end of the algorithm
- ☐ at the end of NDFS algo[1]. every state is blue or an accepting cycle has been found
- ☐ ndfs() reports an accepting cycle if and only if one is reachable from the initial state

[1] If NDFS terminates without reporting a cycle

Table of contents

Presentation of F*

- F* is a general-purpose proof-oriented programming language
- dependent types
- proof automation based on SMT solving (based on Z3)
- tactic-based interactive theorem proving
- `Steel` library (or Pulse) allows us to prove concurrent programs

Example: dfs-red in F* (work in progress)

```
1  let rec dfs_red' (g: graph) (s: state) (l: list state) (e: env)
2      : Dv env
3      = match l with
4      | [] -> e
5      | s' :: todo ->
6          if is_cyan s' e
7          then set_result e
8          else
9              let e = dfs_red' g s todo e
10             in let e = dfs_red g s' l e
11             in e
12 and dfs_red (g: graph) (s: state) (l: list state) (e: env)
13     : Dv env
14     = let e' = put_red s e
15     in dfs_red' g s (g.succ s) e'
```

Table of contents

Principle of CNDFS:

- launch several instances of NDFS
- randomly exploring the graph
- and synchronising through shared variables

Issues:

- deal with threads
- deal with randomness
- deal with deadlocks

What was done and what remains to be done

- intermediate proofs for dfs_red in VerCors
- intermediate proofs for dfs_blue in verCors
- NDFS written in F*

- (FW) complete NDFS proof in VerCors
- (FW) continue investigation with F*
- (long term FW) from NDFS to CNDFS

Thank you