Generic Program Querying of Higher-Order Languages

Jens Nicolay

Software Languages Lab

Vrije Universiteit Brussel

Idea

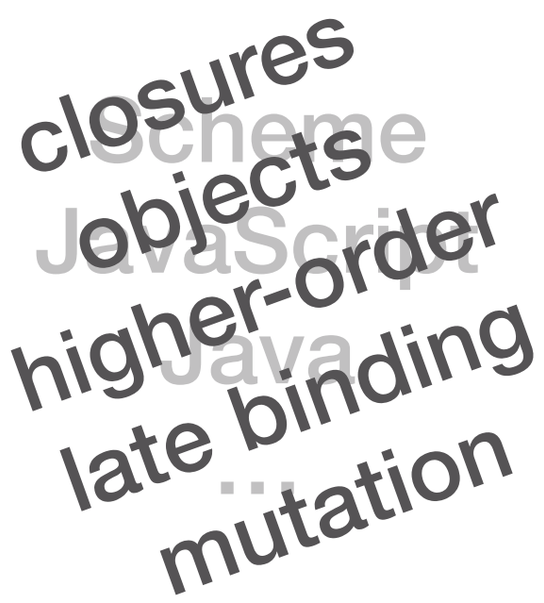
SE tools require static analysis

Scheme
JavaScript
Java
...

liveness
dependence
purity
constant propagation
...

Idea

SE tools require static analysis



common ground

Idea

Don't start from scratch

Plan

```
Pushdown.pathsBwTo =
  function (s, target, etg)
  {
    var todo = [s];
    var visited = ArraySet.empty();
    var paths = ArraySet.empty();
    while (todo.length > 0)
    {
      var q = todo.shift();
      if (q.equals(target) || visited.contains(q))
      {
        continue;
      }
      visited = visited.add(q);
      var incoming = etg.incoming(q)
      paths = paths.addAll(incoming);
      var qs = incoming.map(Edge.source);
      todo = todo.concat(qs);
    }
    return paths.values();
  }
```

Plan

```
(letrec ((factor (lambda (n)
                   (let ((extract-twos
                          (lambda (n)
                            (letrec ((loop (lambda (two-list rest)
                                             (if (even? rest)
                                                 (loop (cons 2 two-list) (quotient rest 2))
                                                 (cons rest two-list)))))
                              (loop '() n)))))
                     (let ((extract-odd-factors
                            (lambda (partial-factorization)
                              (letrec ((loop (lambda (so-far odd-product trial-divisor)
                                               (if (< odd-product (* trial-divisor trial-divisor))
                                                   (reverse (cons odd-product so-far))
                                                   (if (= (remainder odd-product trial-divisor) 0)
                                                       (loop (cons trial-divisor so-far)
                                                             (quotient odd-product trial-divisor)
                                                             trial-divisor)
                                                       (loop so-far
                                                             odd-product
                                                             (+ trial-divisor 2)))))))
                                (loop (cdr partial-factorization) (car partial-factorization) 3)))))
                       (let ((partial-factorization (extract-twos n)))
                         (if (= (car partial-factorization) 1)
                             (cdr partial-factorization)
                             (extract-odd-factors partial-factorization))))))))
  (factor 35742549198872617291))
```

```
Pushdown.pathsBwTo =
  function (s, target, etg)
  {
    var todo = [s];
    var visited = ArraySet.empty();
    var paths = ArraySet.empty();
    while (todo.length > 0)
    {
      var q = todo.shift();
      if (q.equals(target) || visited.contains(q))
      {
        continue;
      }
      visited = visited.add(q);
      var incoming = etg.incoming(q)
      paths = paths.addAll(incoming);
      var qs = incoming.map(Edge.source);
      todo = todo.concat(qs);
    }
    return paths.values();
  }
```

```
public void prune(DirectedGraph<Vertex, Edge> g)
{
  Deque<Vertex> todo = new ArrayDeque<Vertex>(g.vertexSet());
  Set<Object> seen = new HashSet<Object>();
  while (!todo.isEmpty())
  {
    Vertex v = todo.pop();
    if (seen.contains(v))
    {
      continue;
    }
    seen.add(v);
    List<Vertex> targets = Graphs.successorListOf(g, v);
    for (int i = 0; i < targets.size(); i++)
    {
      for (int j = i; j < targets.size(); j++)
      {
        Vertex vi = targets.get(i);
        Vertex vj = targets.get(j);
        List<Edge> path = DijkstraShortestPath.findPathBetween(g, vi, vj);
        if (path != null && !path.isEmpty())
        {
          Edge edge = g.getEdge(v, vj);
          g.removeEdge(edge);
          LOGGER.info("removed " + edge + " because of " + path + " for " + v);
        }
      }
    }
  }
}
```

Plan

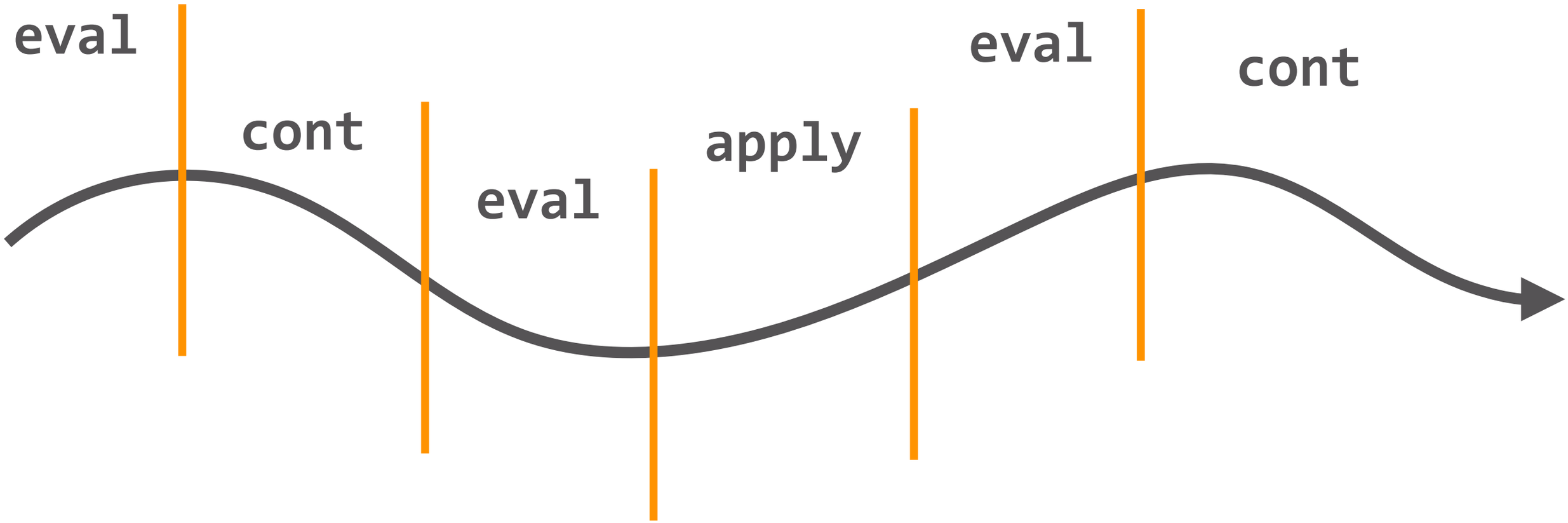
liveness analysis

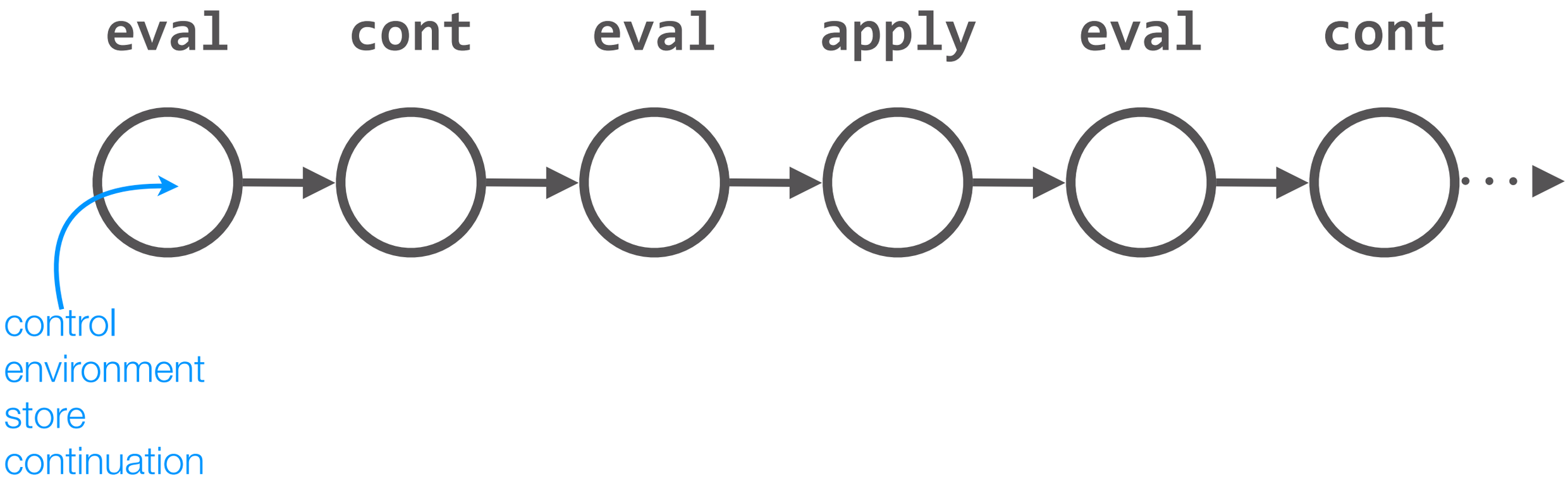
purity analysis

dependence analysis

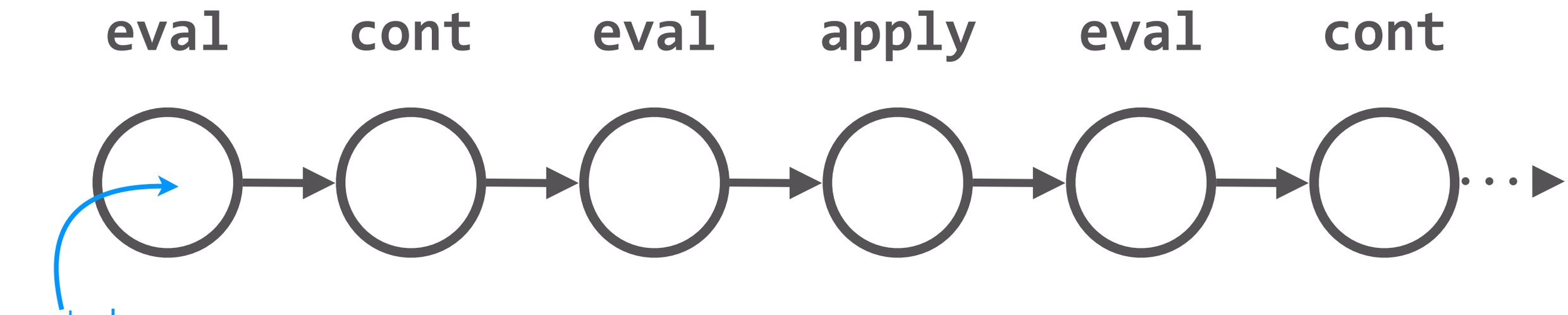
constant propagation

...


eval
cont
eval
apply
eval
cont

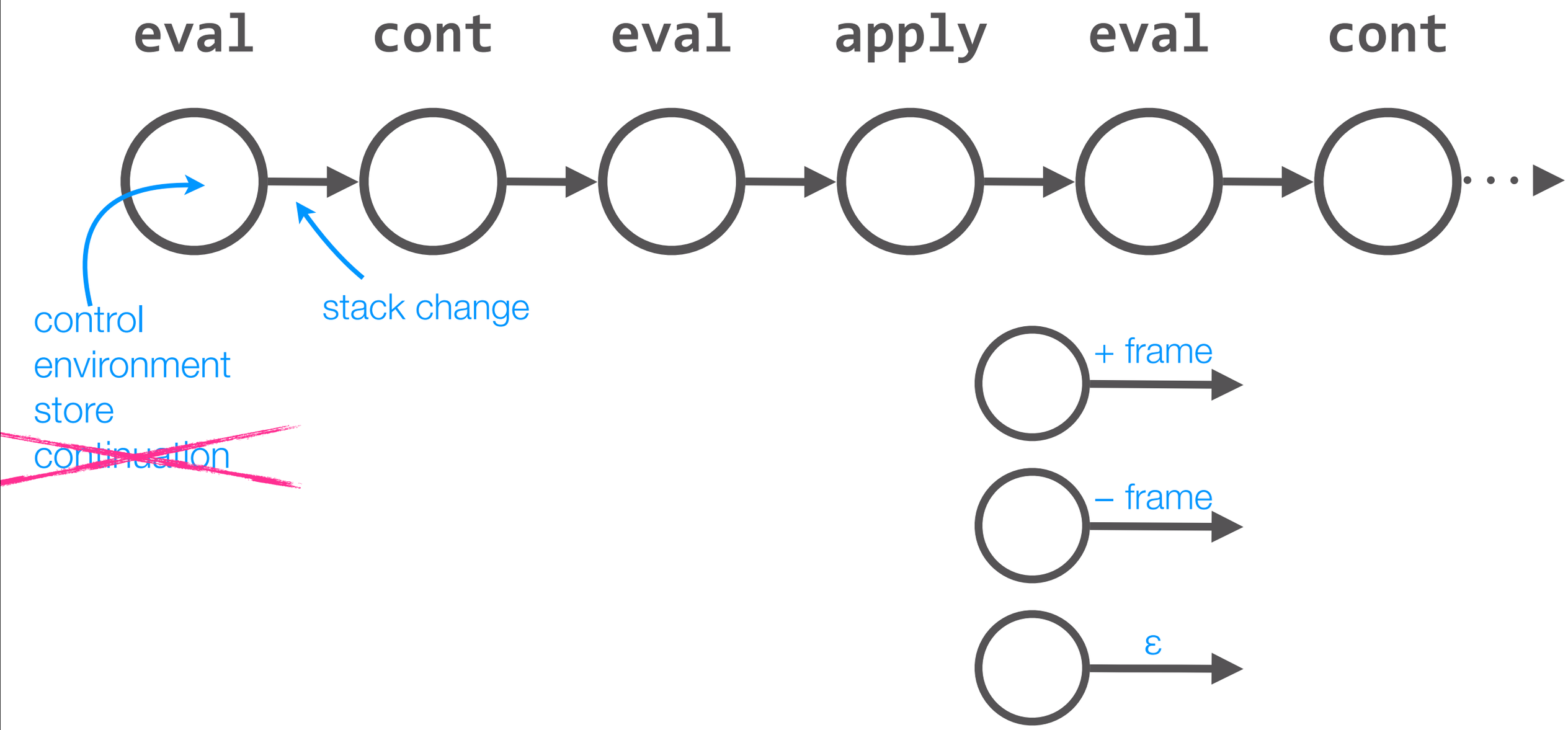

eval
cont
eval
apply
eval
cont
control
environment
store
continuation

Abstract Interpretation



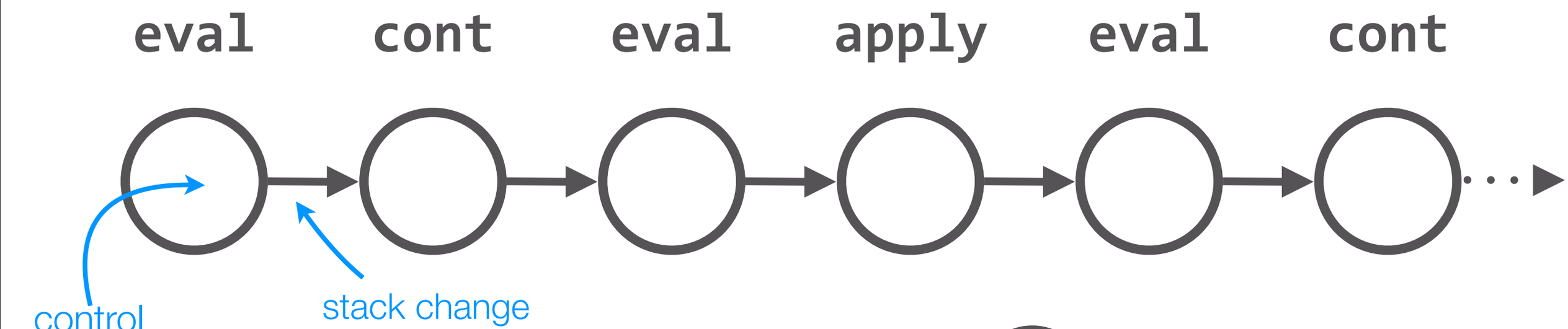
- finite number of abstract values
- finite number of addresses

(Abstracting Abstract Machines, Van Horn and Might, ICFP2010)

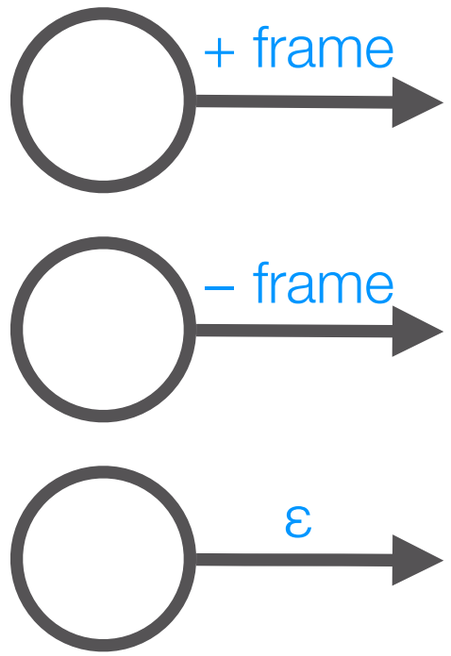


(Pushdown control-flow analysis of higher-order programs, Earl et al., Scheme2010)

Pushdown Analysis

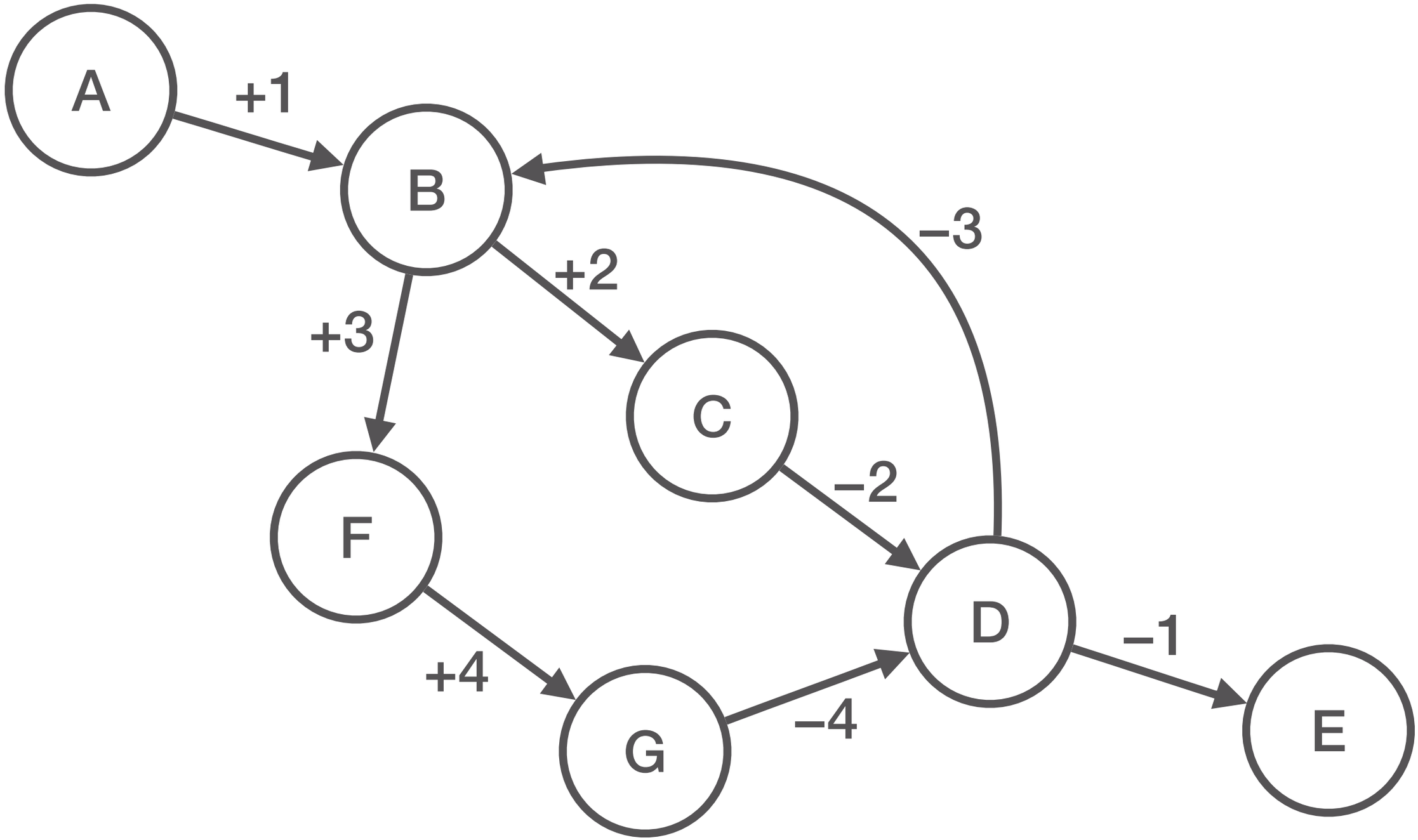


+ pushdown driver

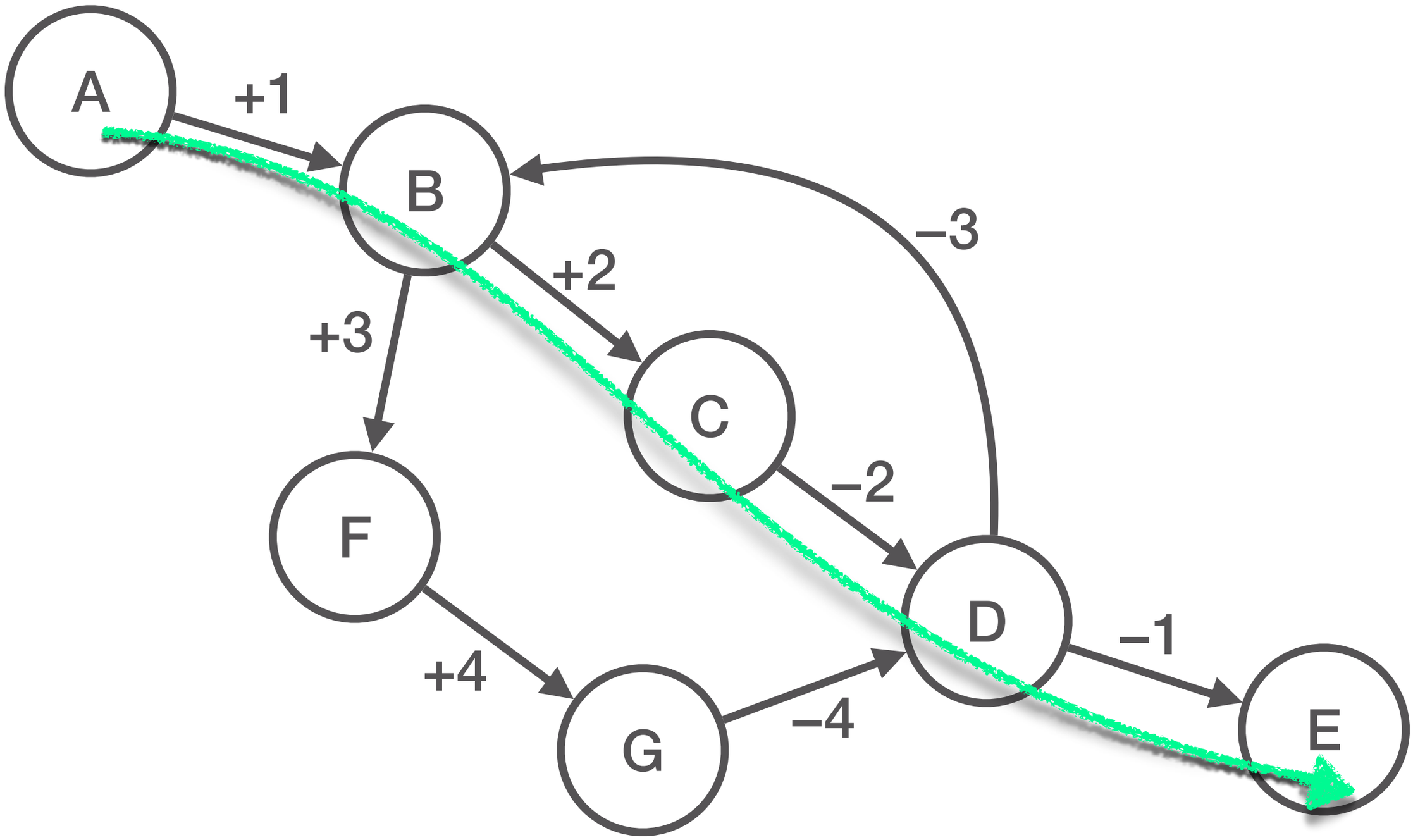


(Pushdown control-flow analysis of higher-order programs, Earl et al., Scheme2010)

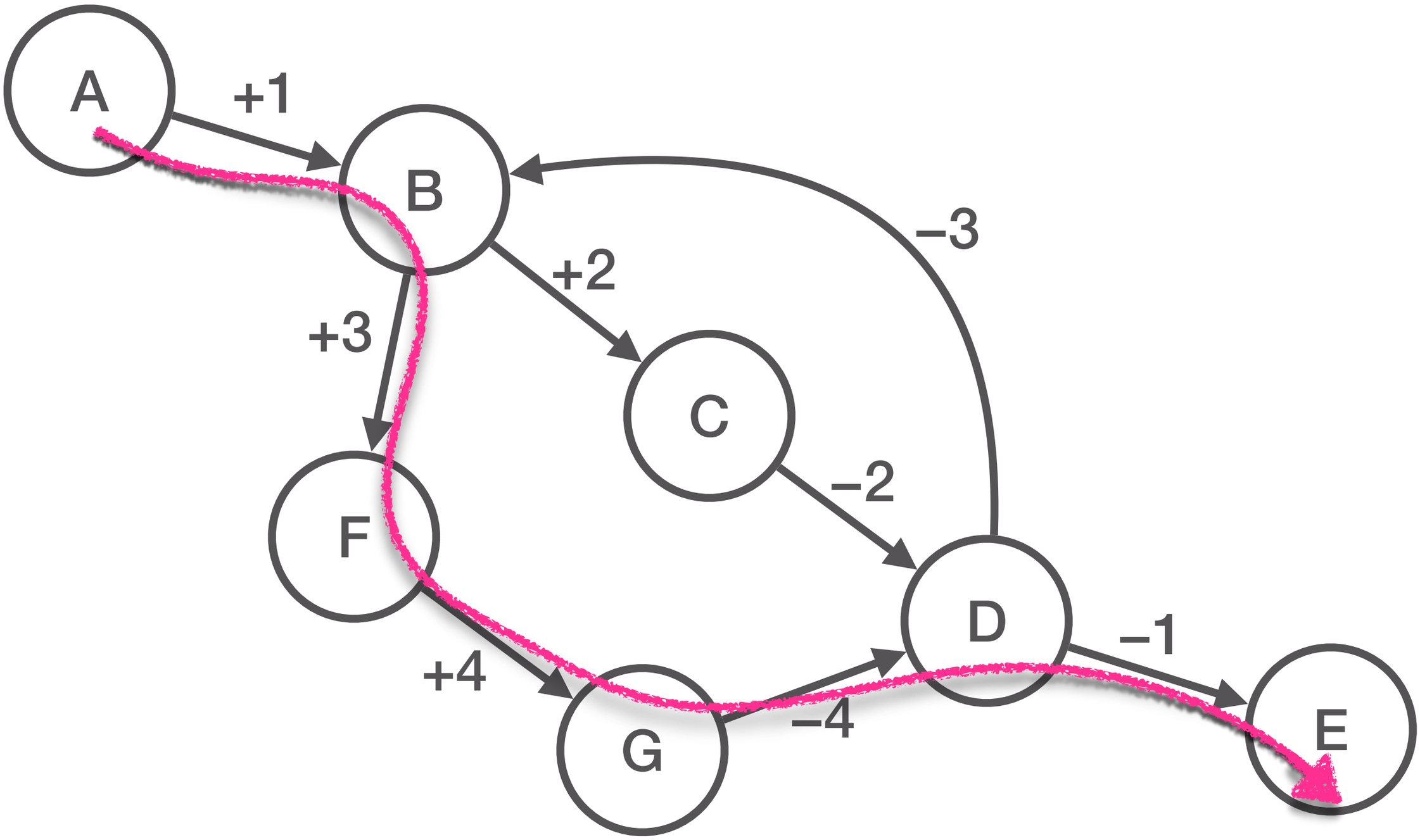
Dyck State Graph



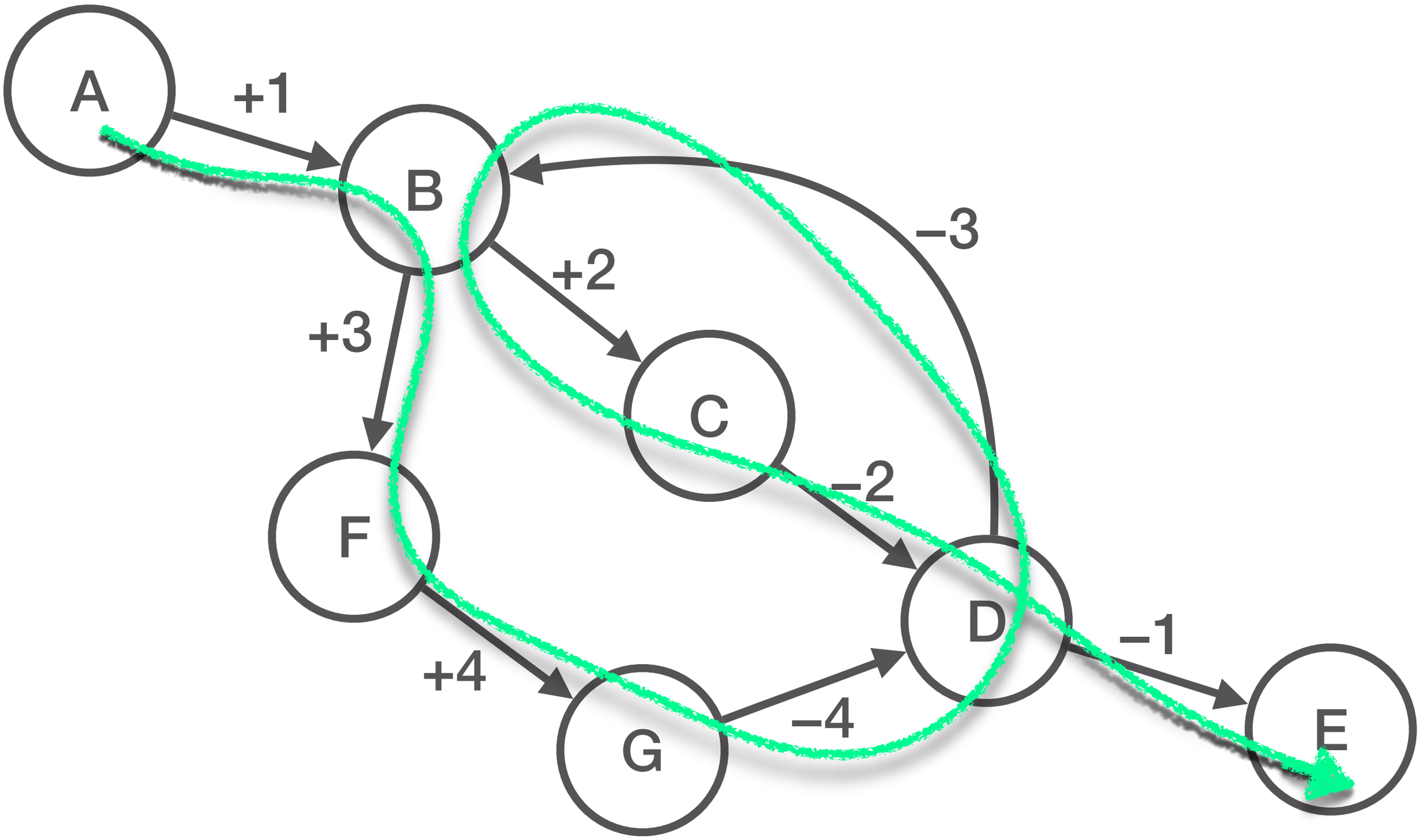
Legal Path



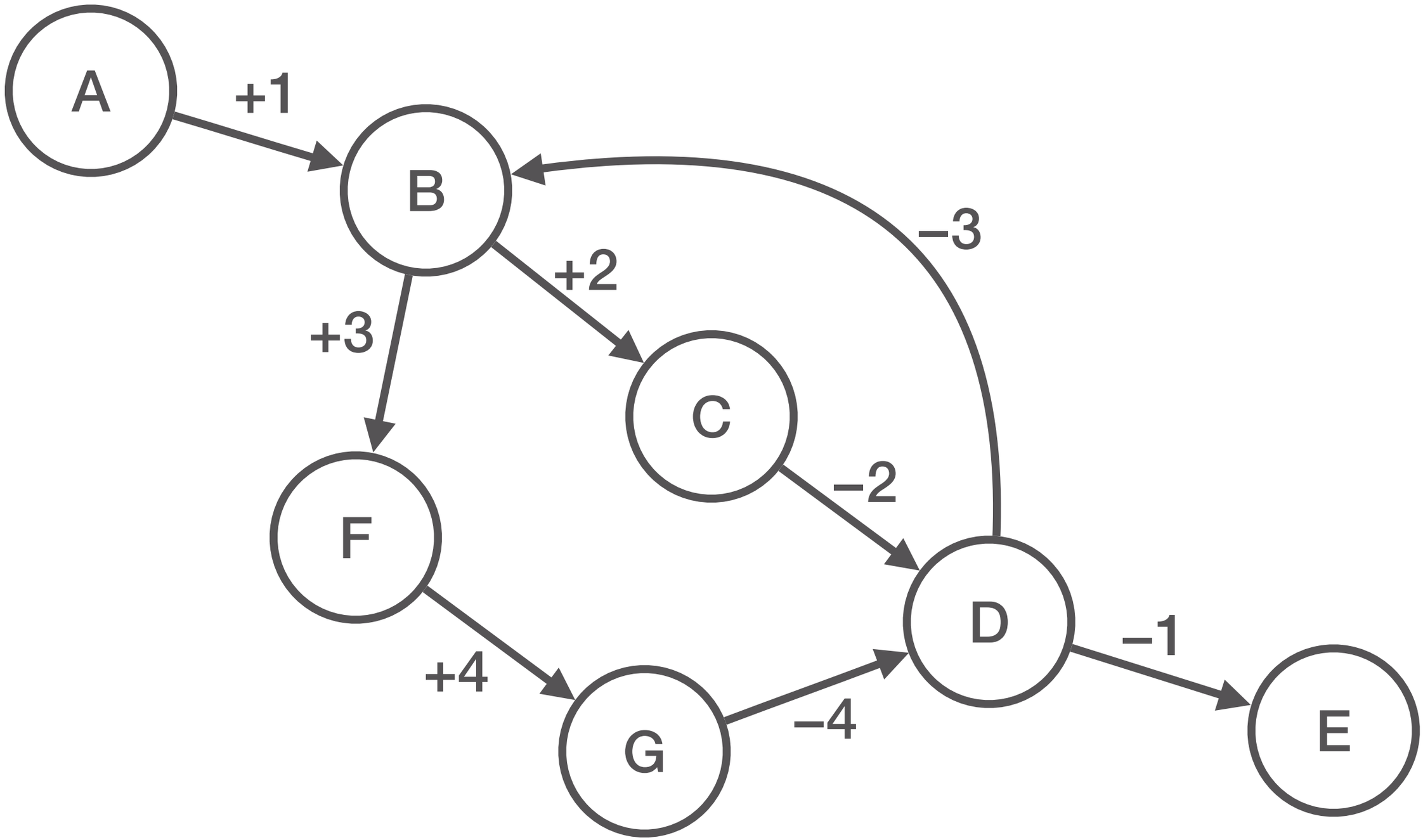
~~Legal~~ Path



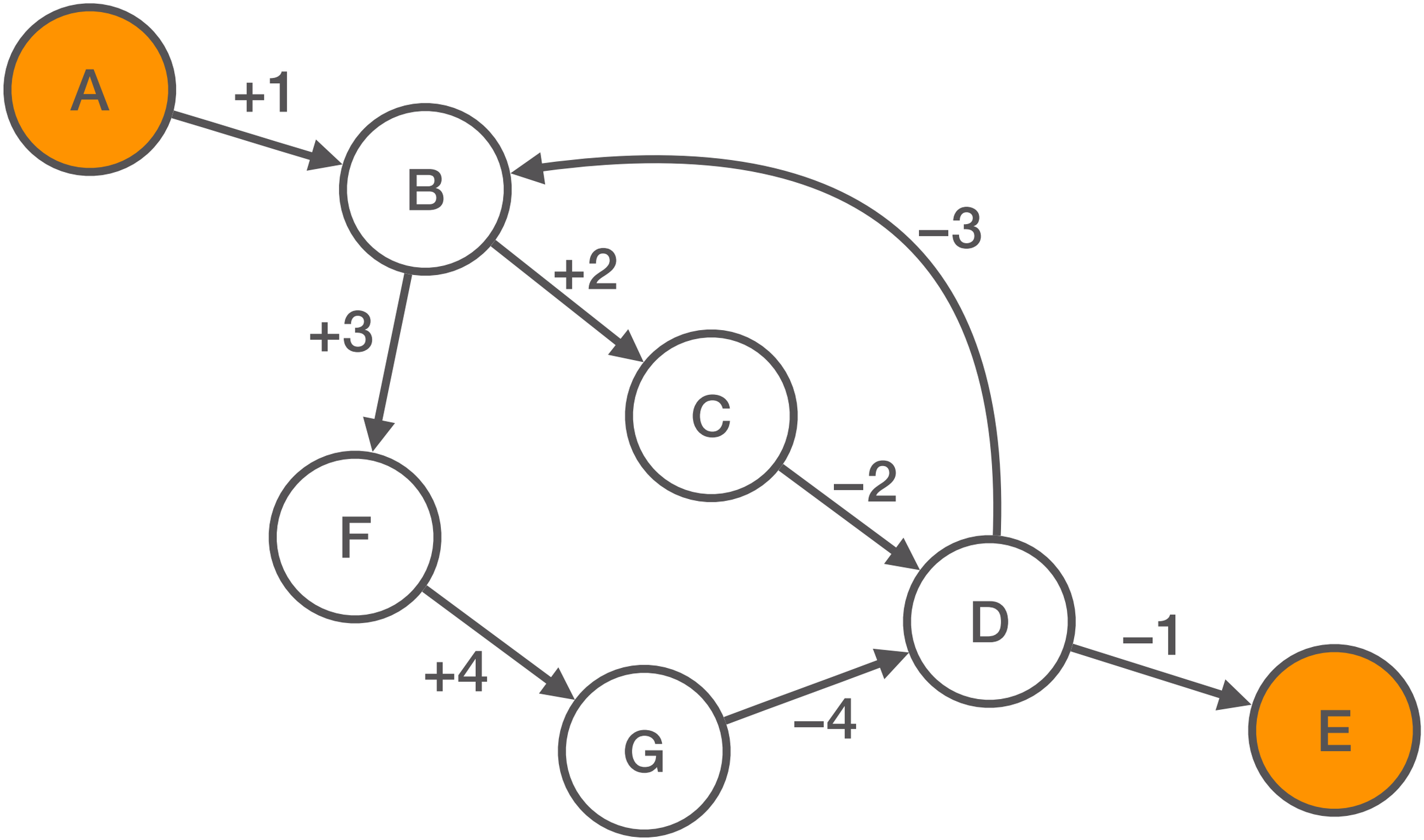
Legal Path



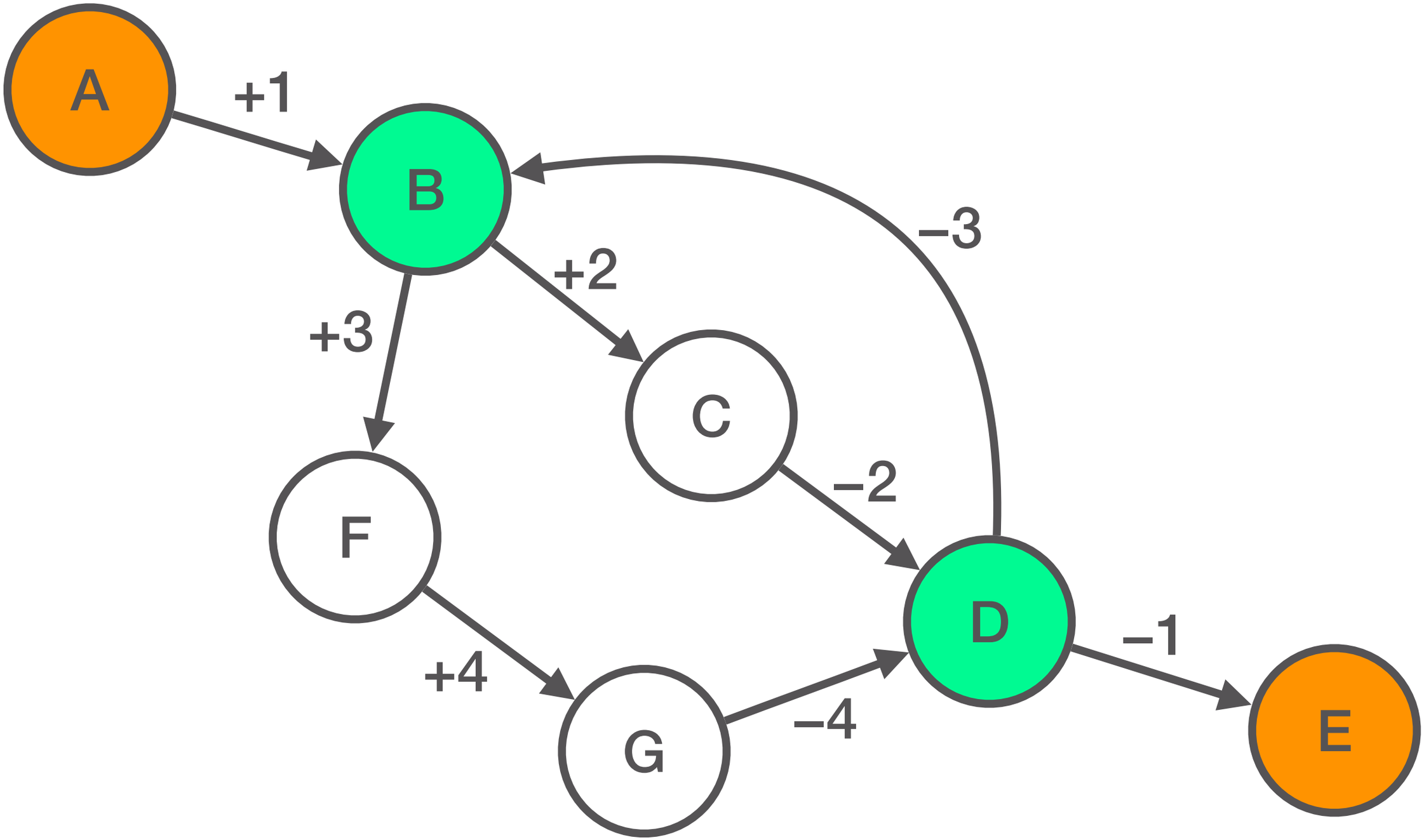
finite representation of unbounded stack



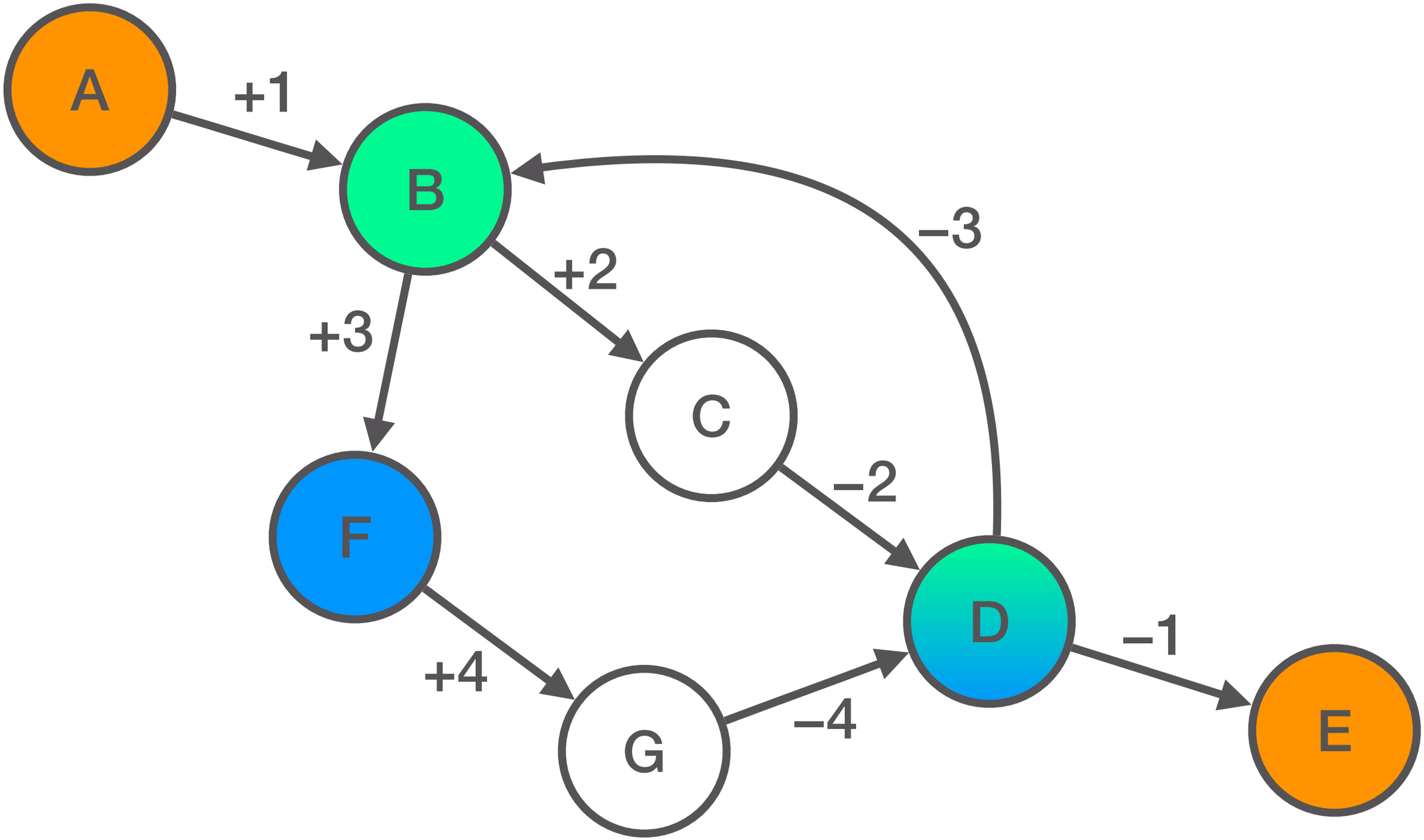
Stack Change



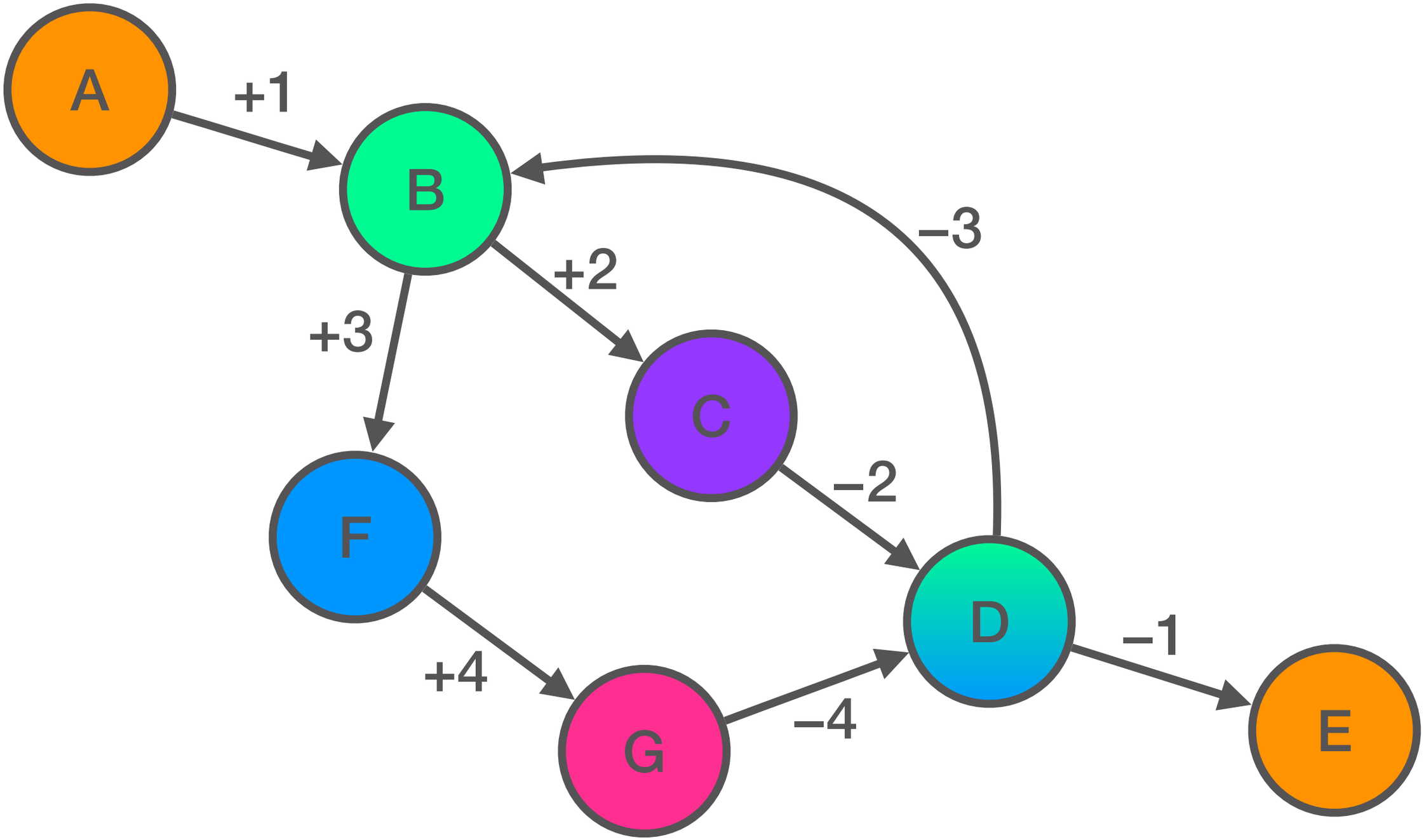
Stack Change



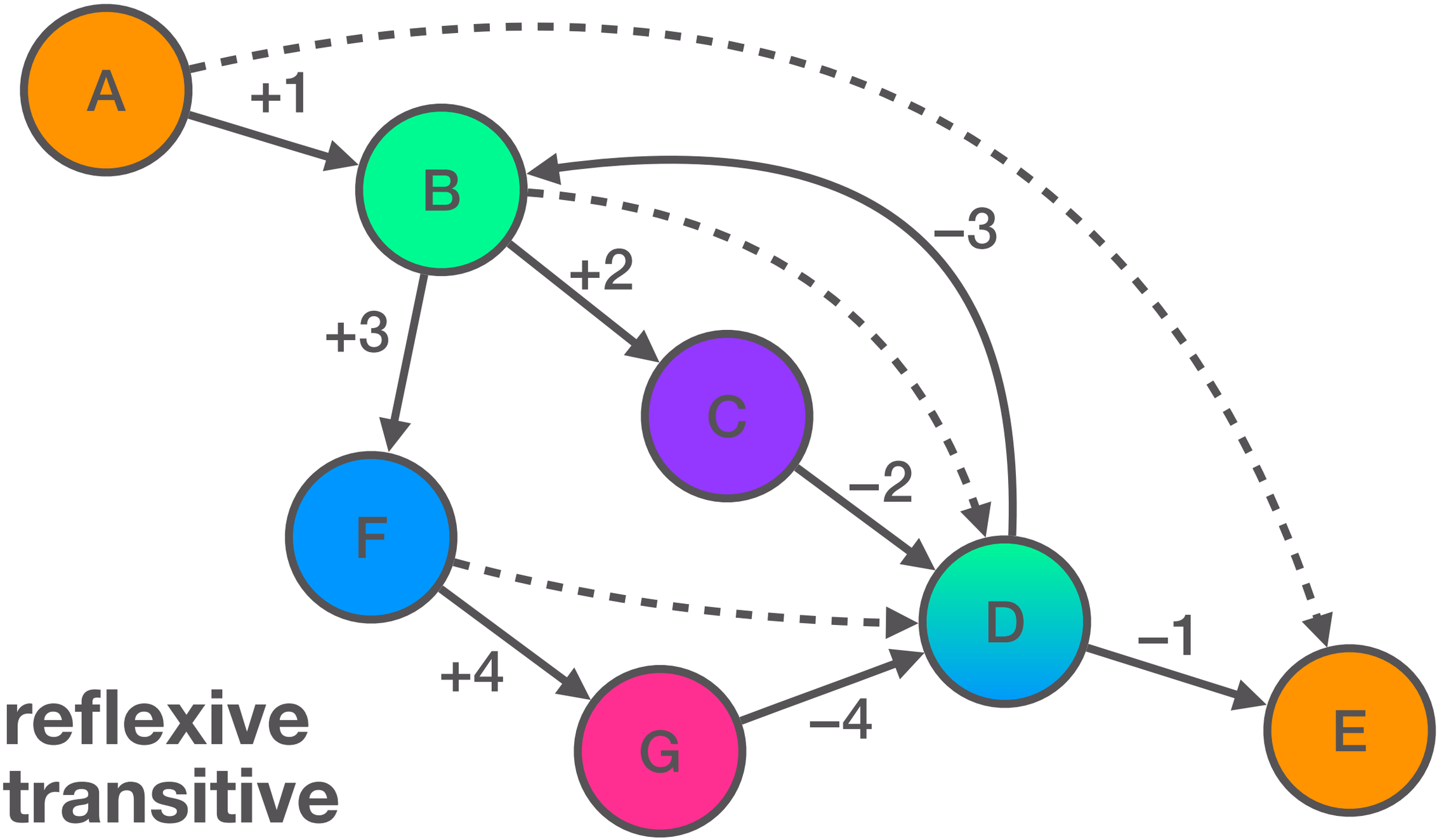
Stack Change



Stack Change

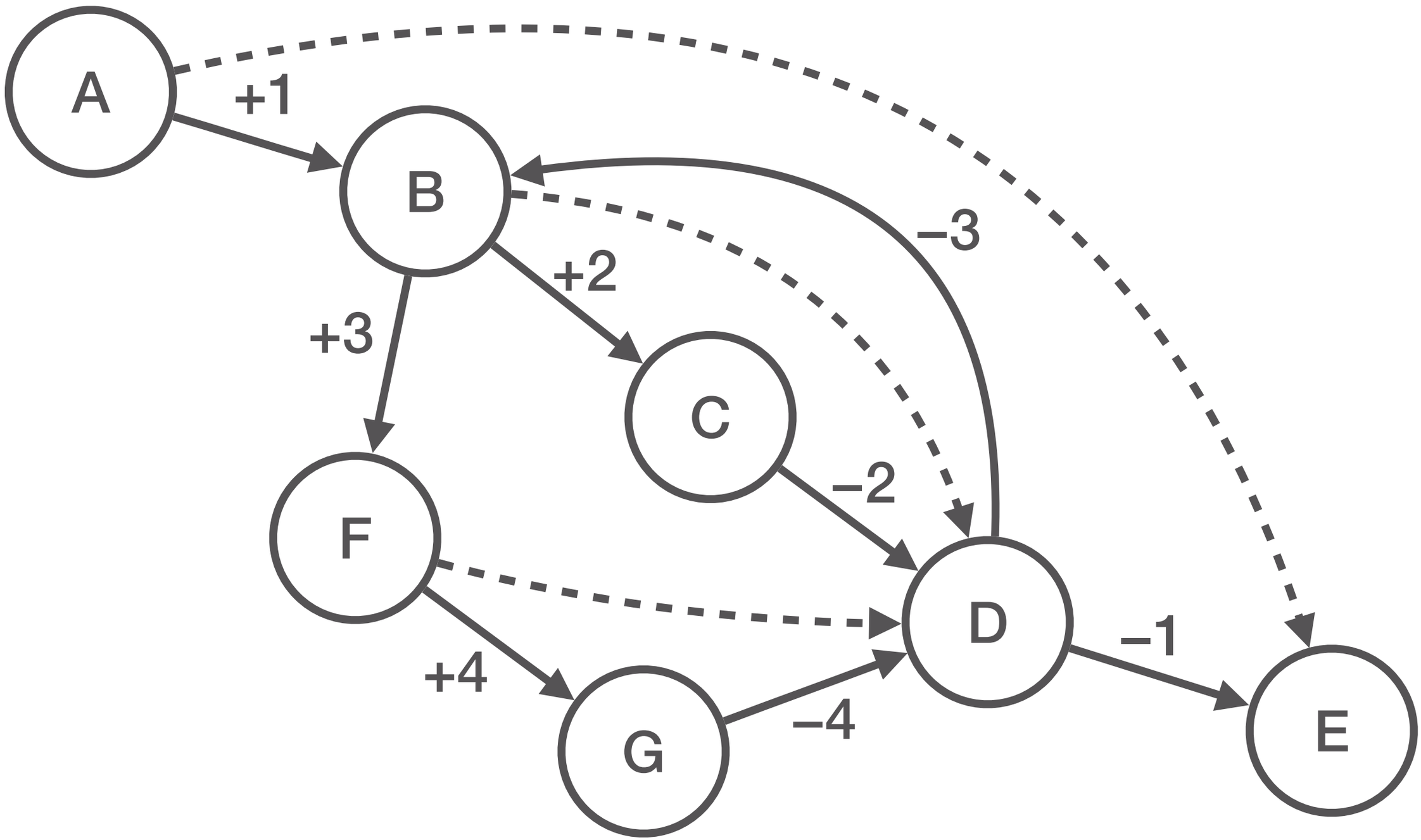


Summary Edges

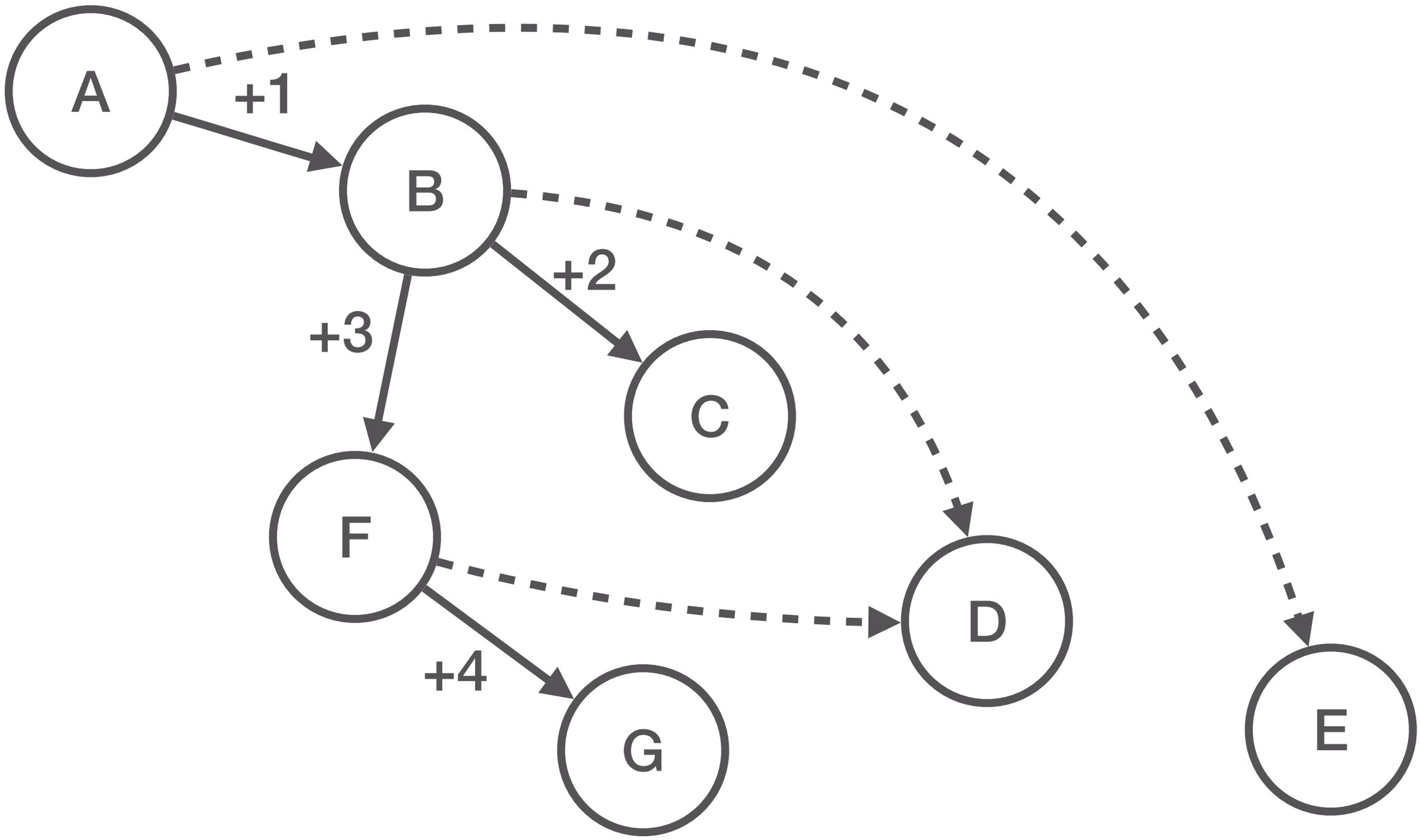


+ reflexive
+ transitive

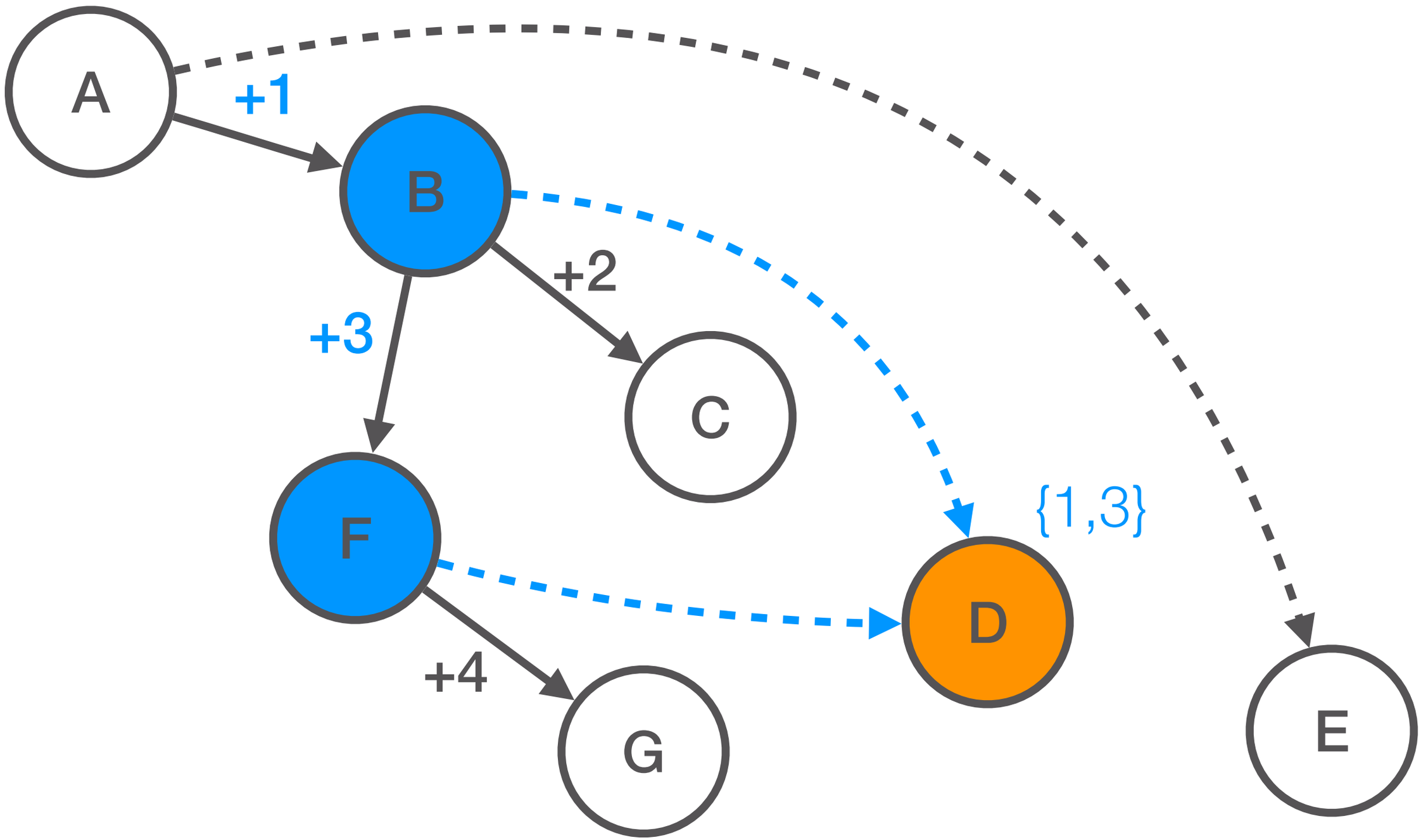
DSG Querying



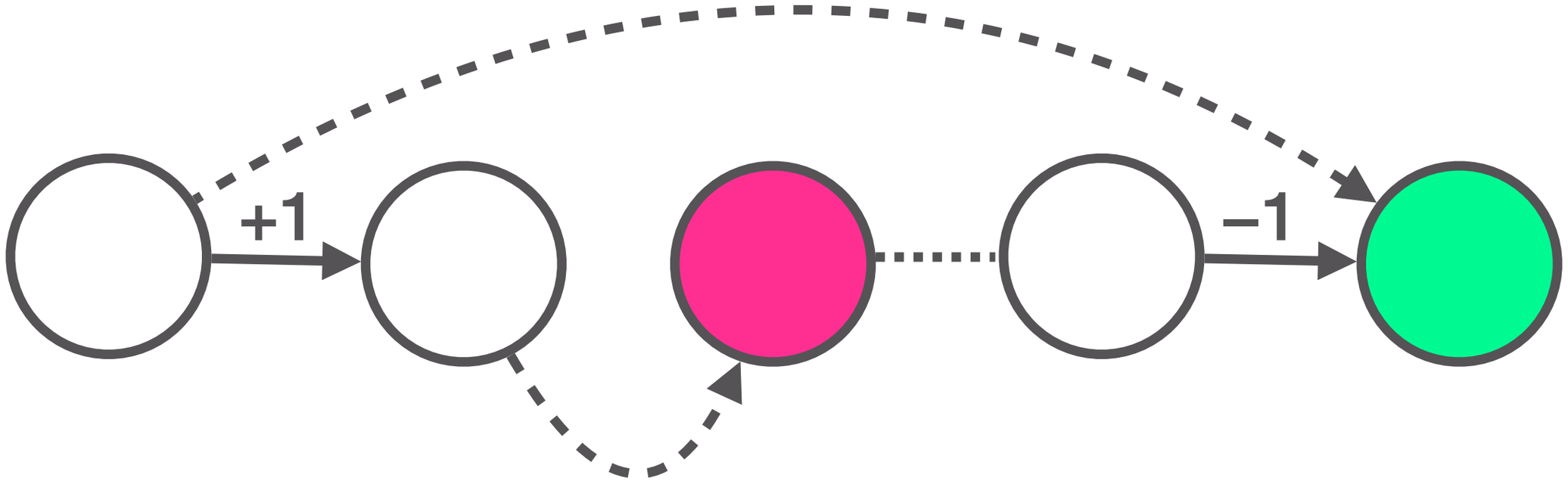
topOfStack



topOfStack



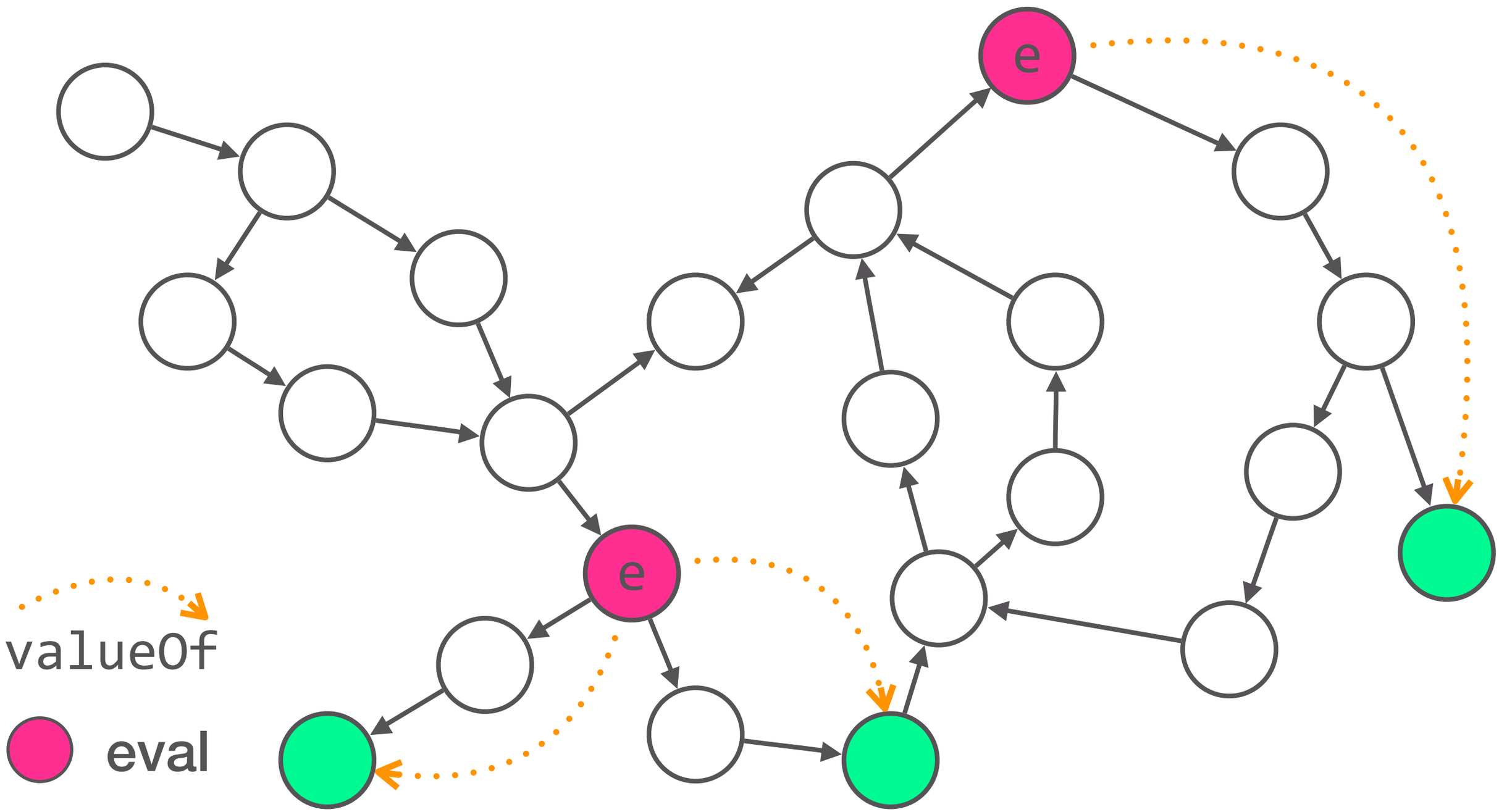
valueOf



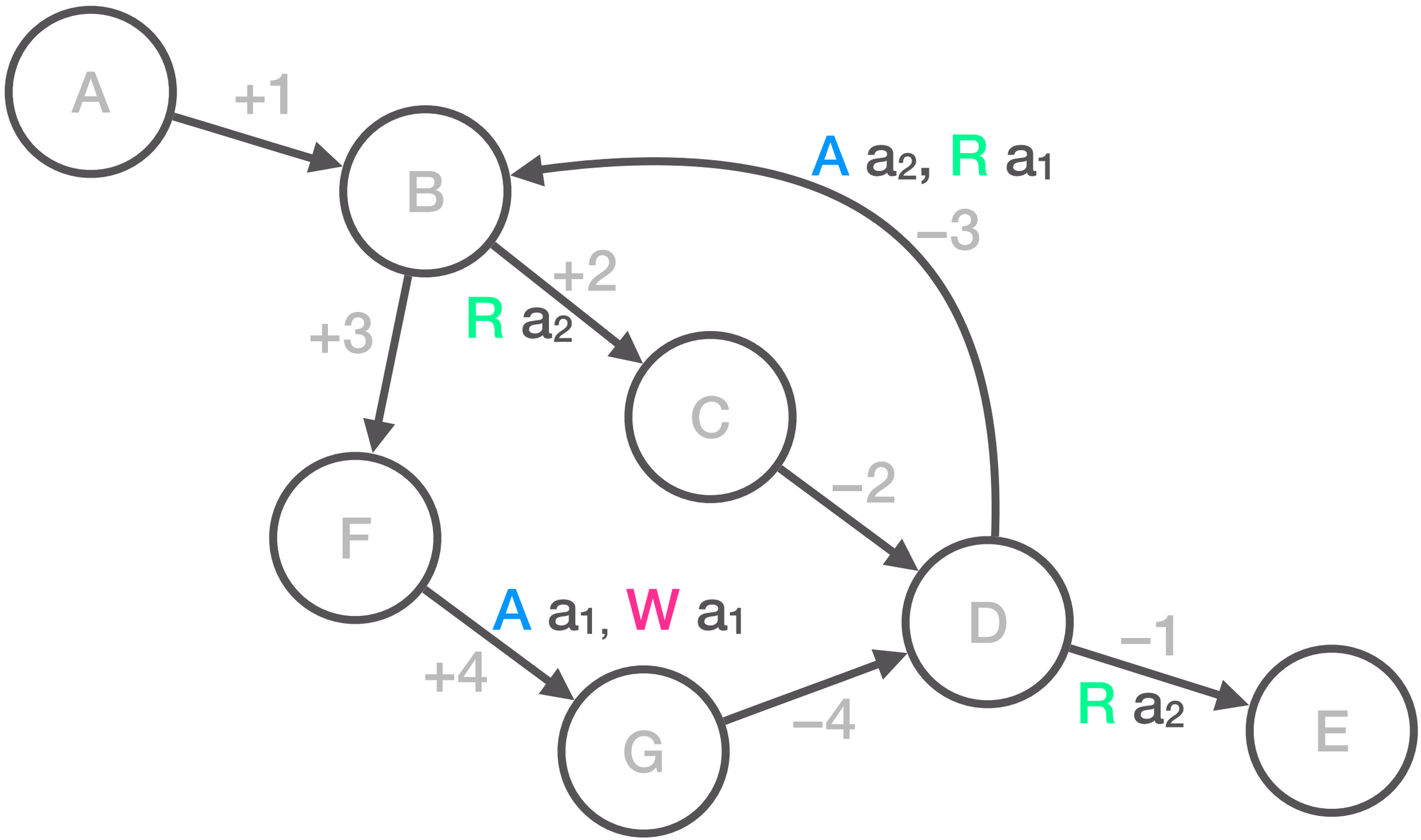
eval

cont

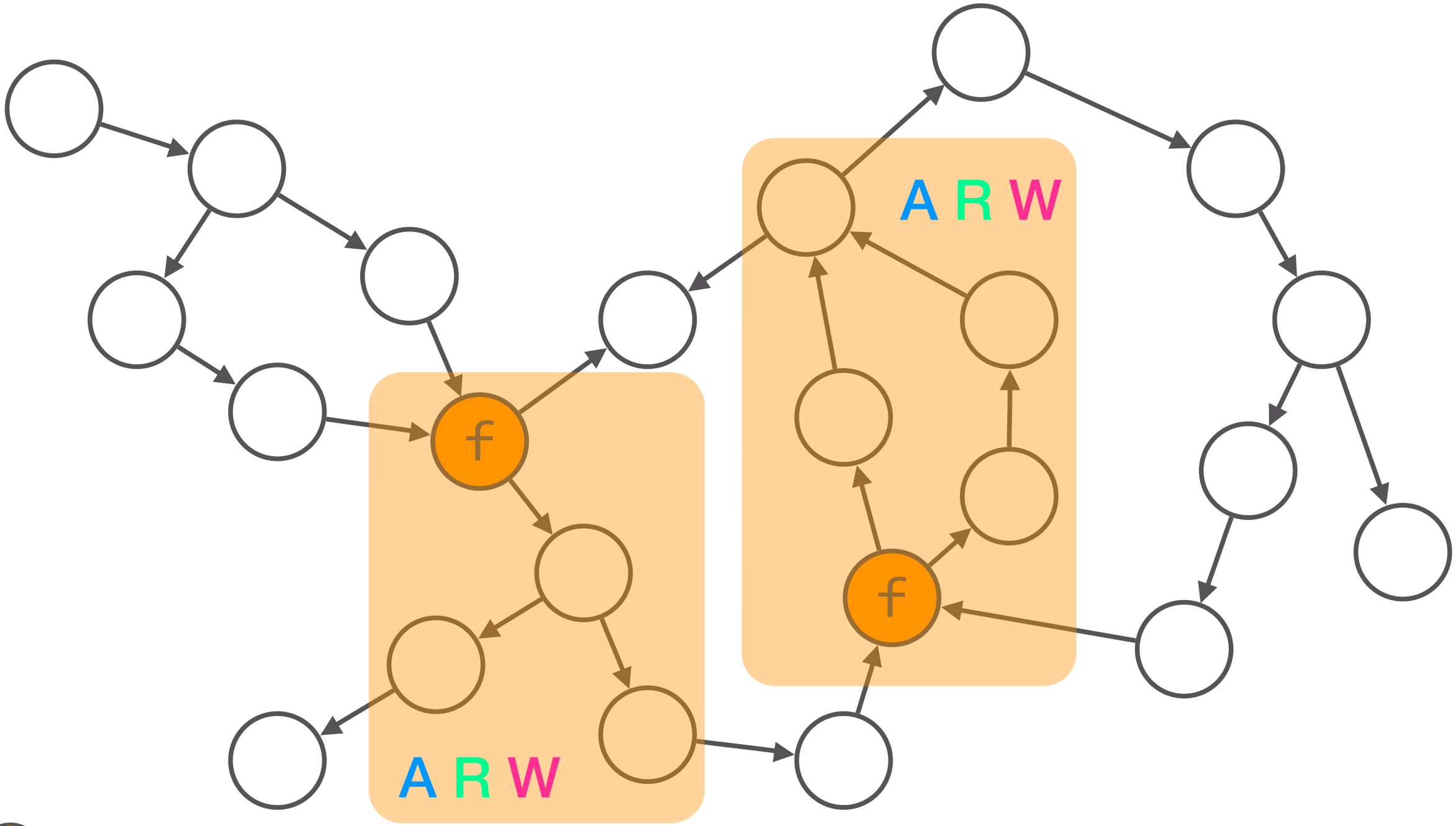
valueOf(e)



Effects



Purity Analysis

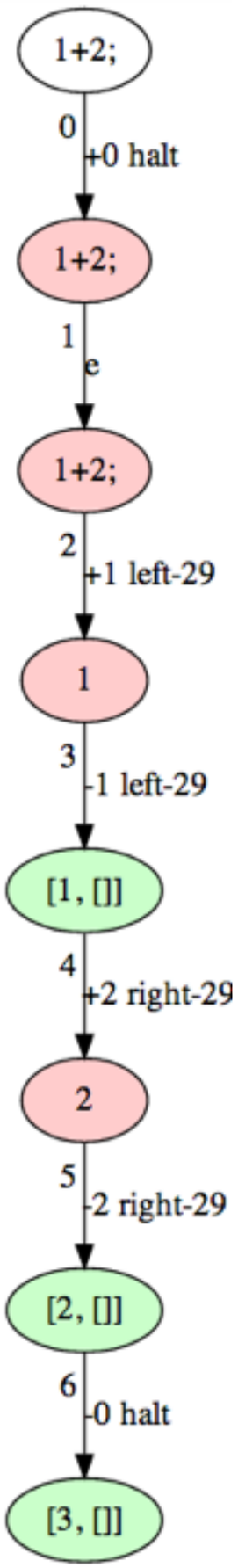


Wrap Up

- build generic pushdown analyses machinery
  - input languages
  - Dyck state graph queries
- find and exploit commonalities
- implementations: `https://github.com/jensnicolay`

Thanks!

1+2


1+2;
0
+0 halt
1+2;
1
e
1+2;
2
+1 left-29
1
3
-1 left-29
[1, []]
4
+2 right-29
2
5
-2 right-29
[2, []]
6
-0 halt
[3, []]