Hardware & Software Verification

John Wickerson

Lecture 6: Formal Hardware Verification
14 November 2024

Introduction

- How can we ensure that a piece of computer hardware is correct?

Introduction

- How can we ensure that a piece of computer hardware is correct?

Introduction

- How can we ensure that a piece of computer hardware is correct?

Introduction

- How can we ensure that a piece of computer hardware is correct?

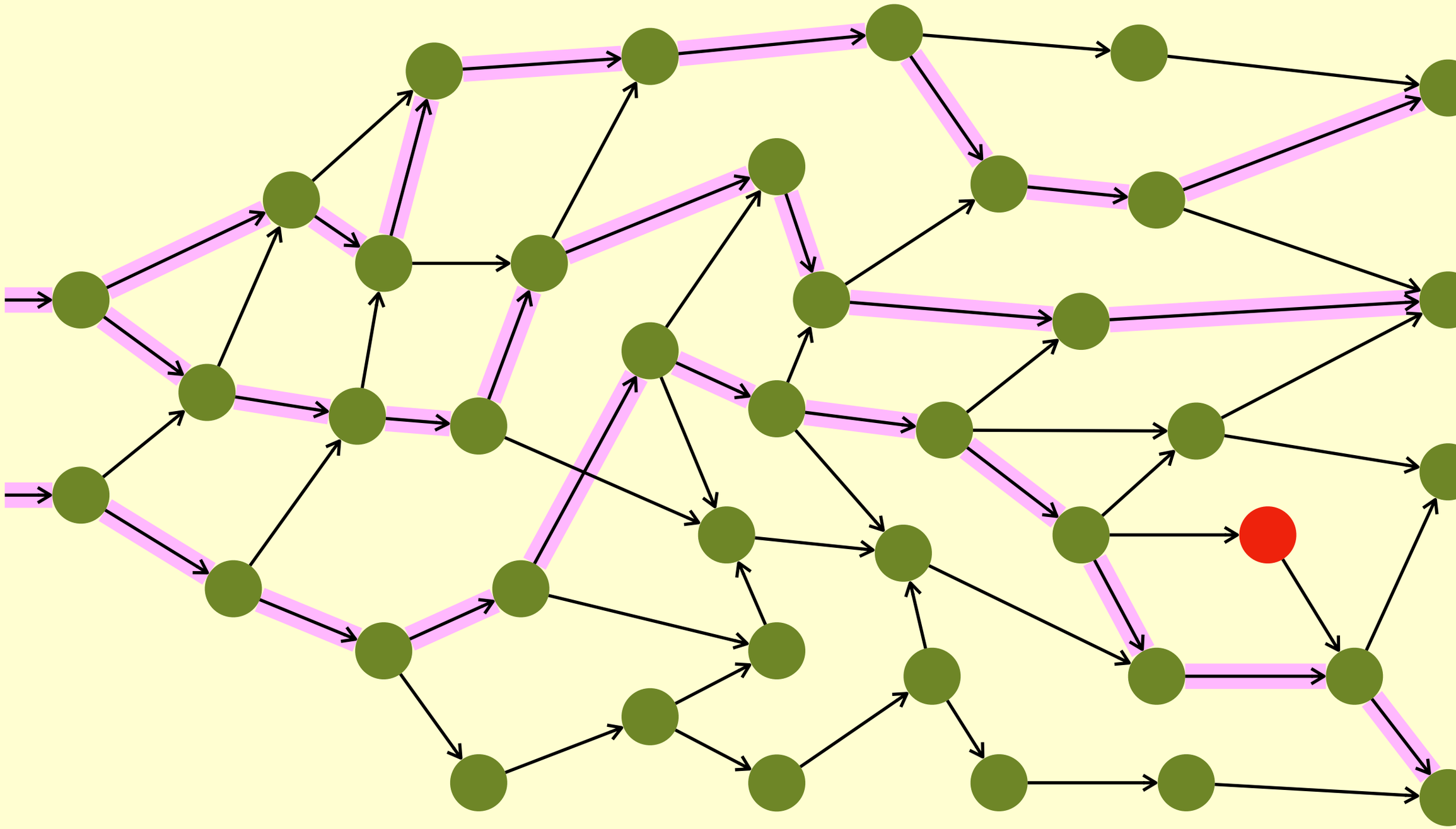
- How could the hardware go wrong?
  - Design faults
  - Fabrication faults
  - Faults during usage

Why verify?

Why verify?

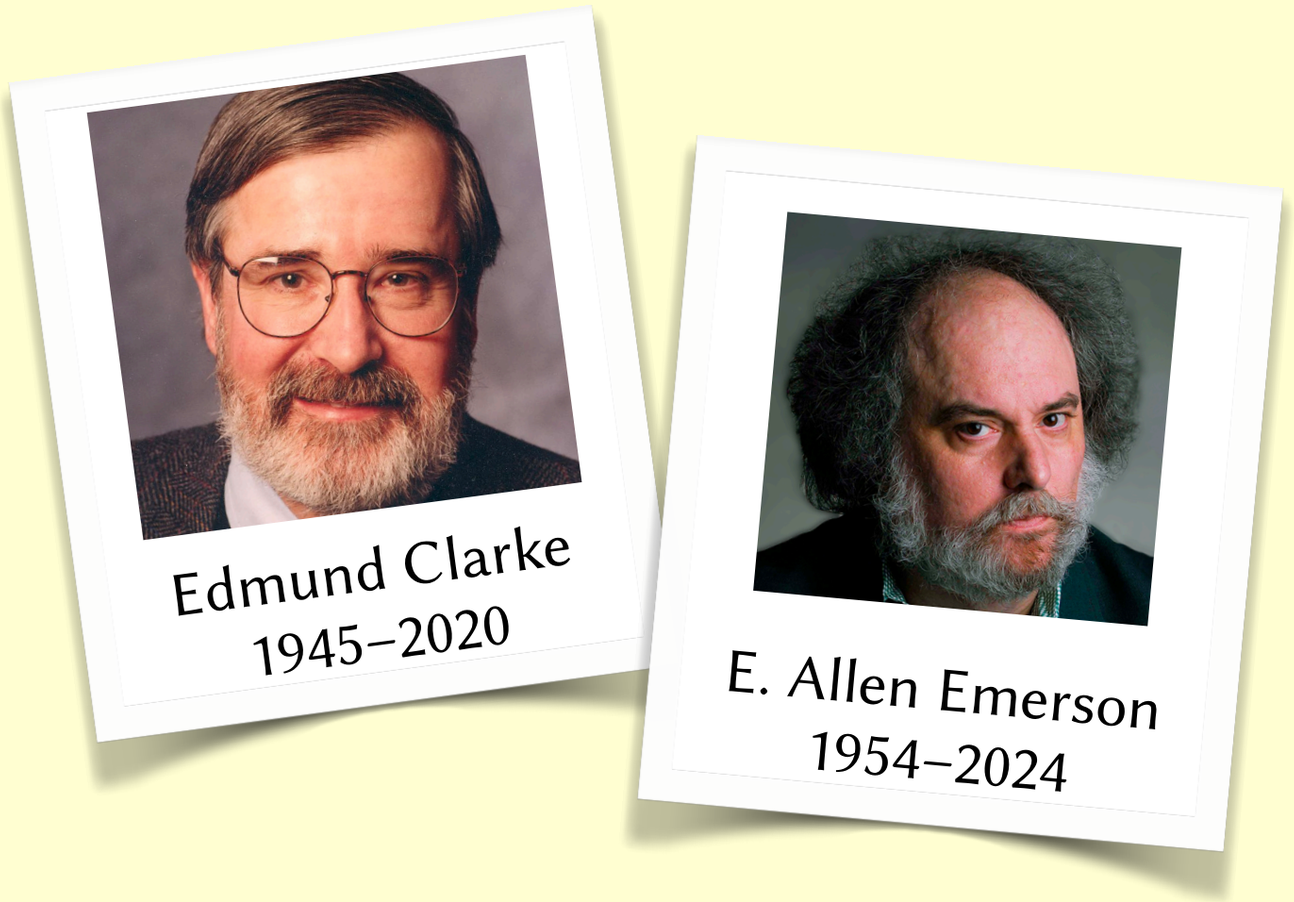

To obtain a perfect design?
To iron out some more bugs
imgflip.com

Simulation vs Formal



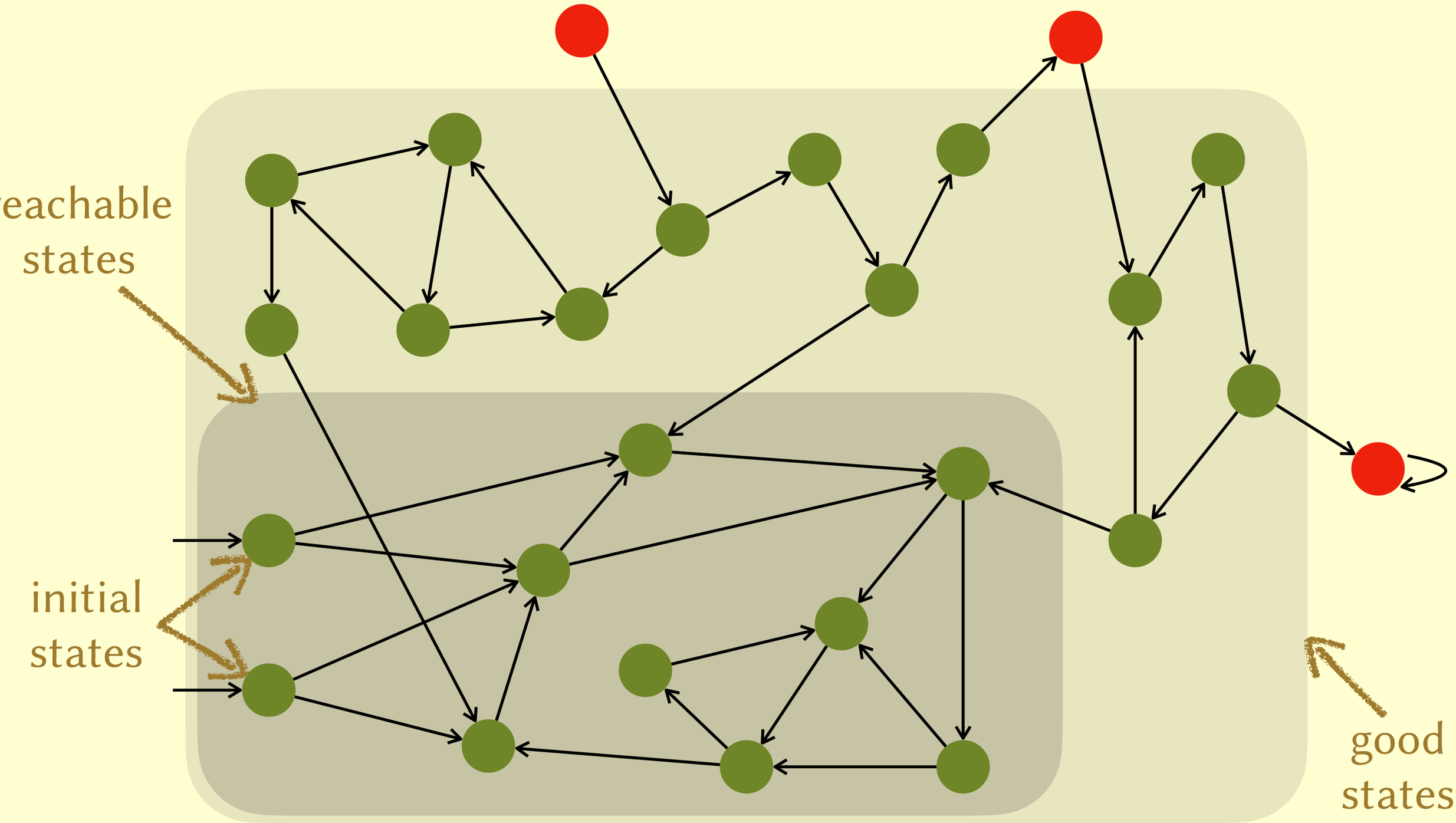
Model checking

- Invented in the early 80s by Clarke and Emerson.



- General idea: automatically prove properties of a finite-state system.

Model checking



Model checking

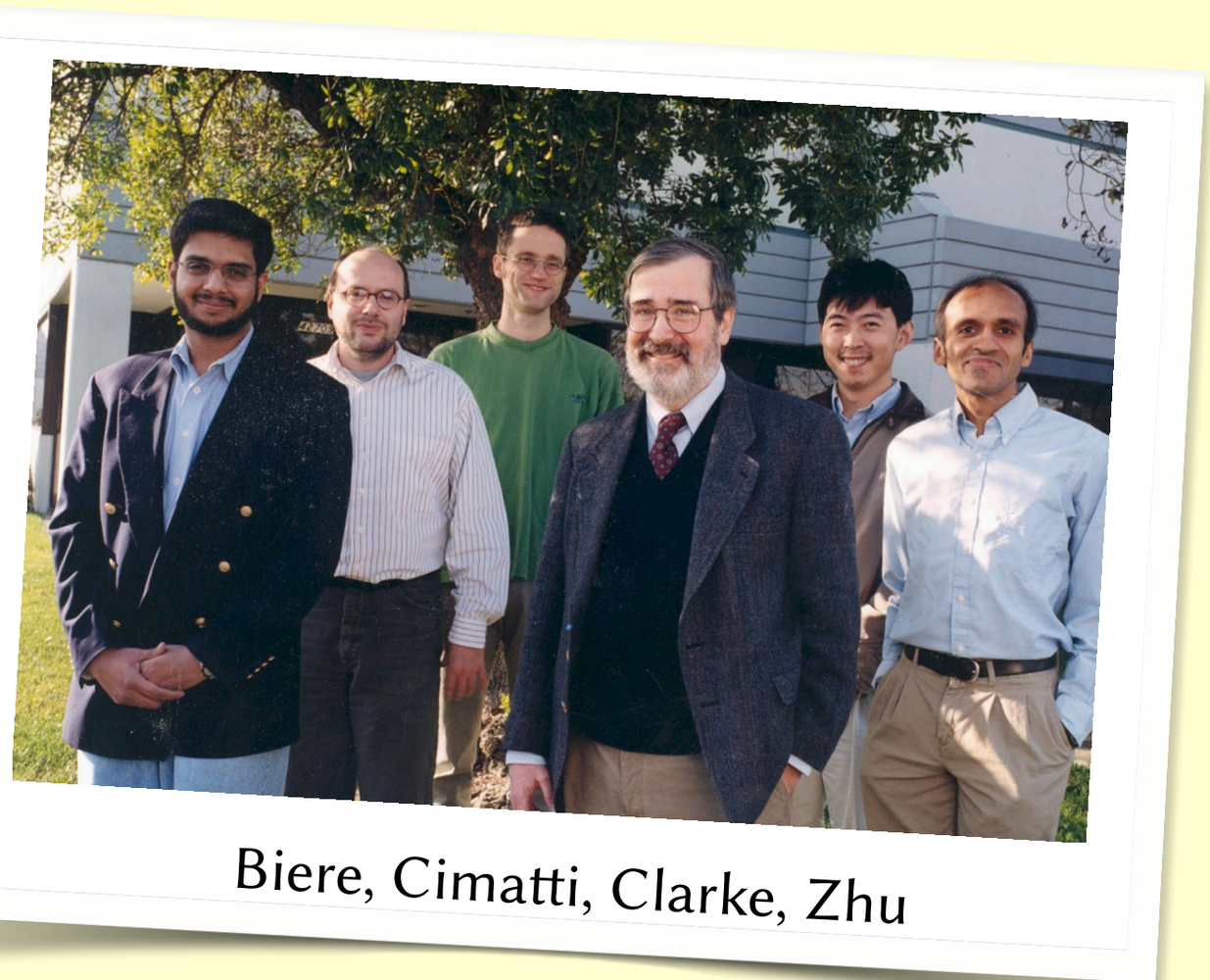
- We want to prove that:

  reachable states ⊆ good states

- When we have a lot of good states that are not reachable, the proof tends to be quite tricky.

- To complete the proof, it is often necessary to define more states to be bad.

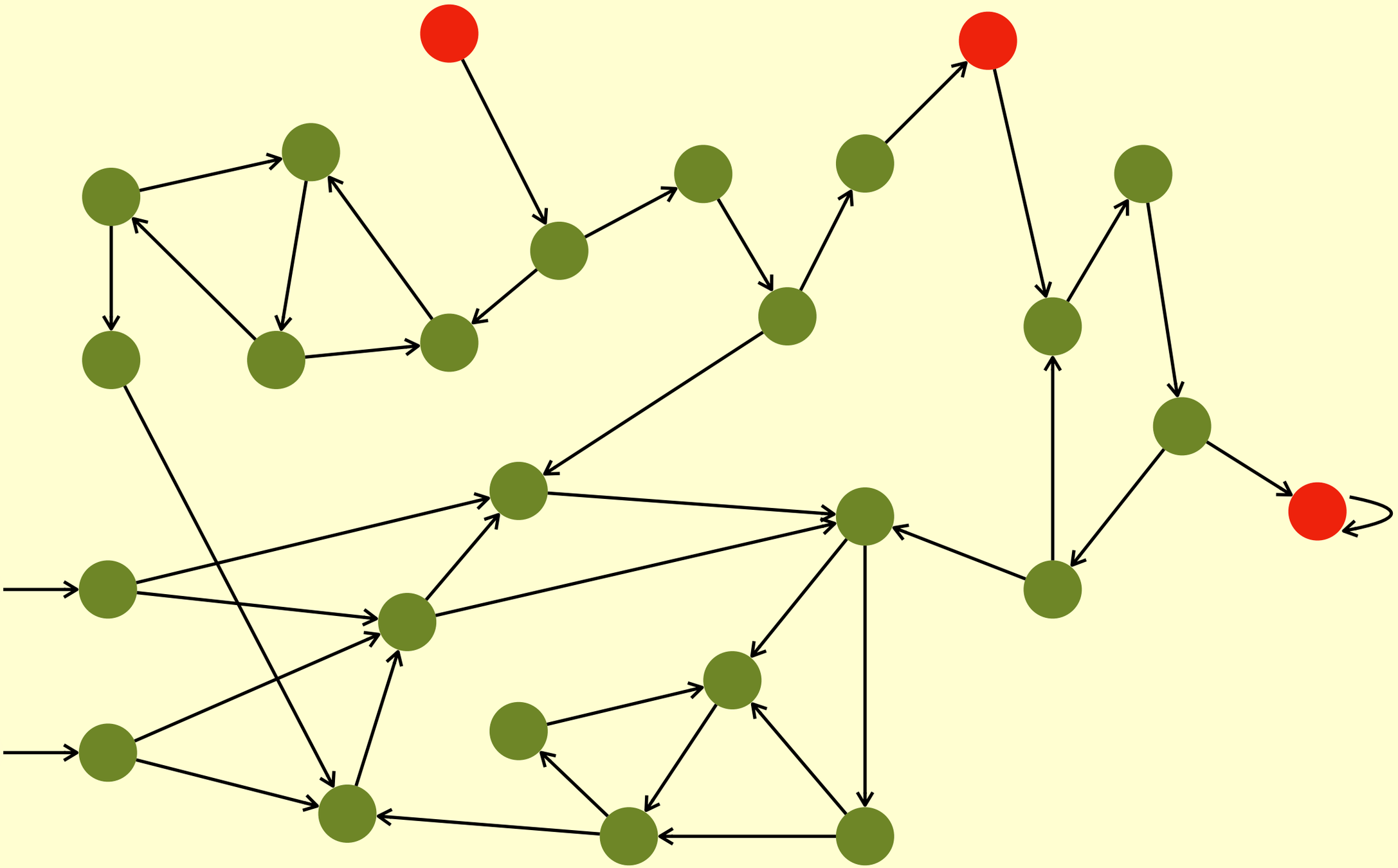
Bounded model checking



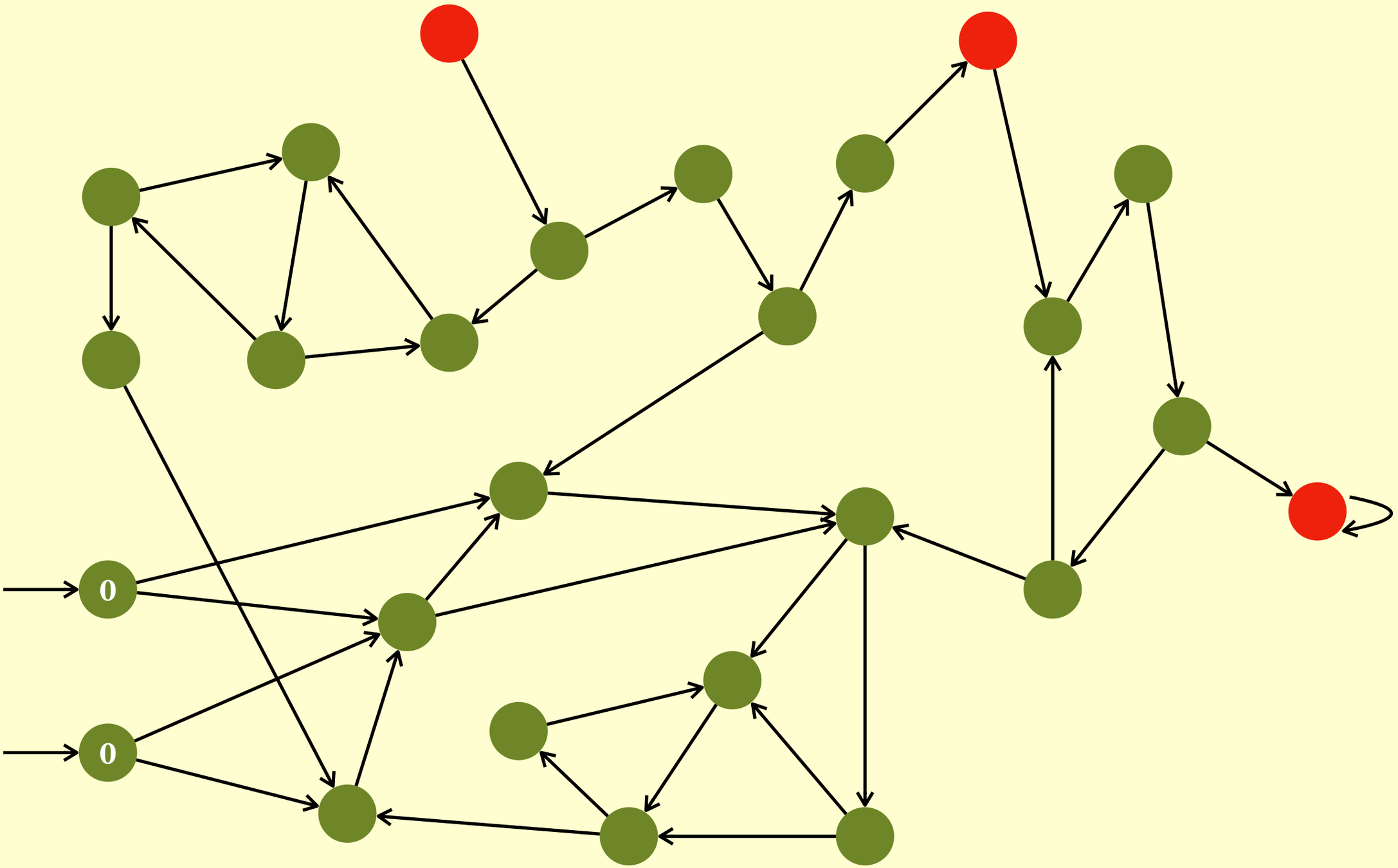
Biere, Cimatti, Clarke, Zhu

- Invented around 1999 by Armin Biere et al.

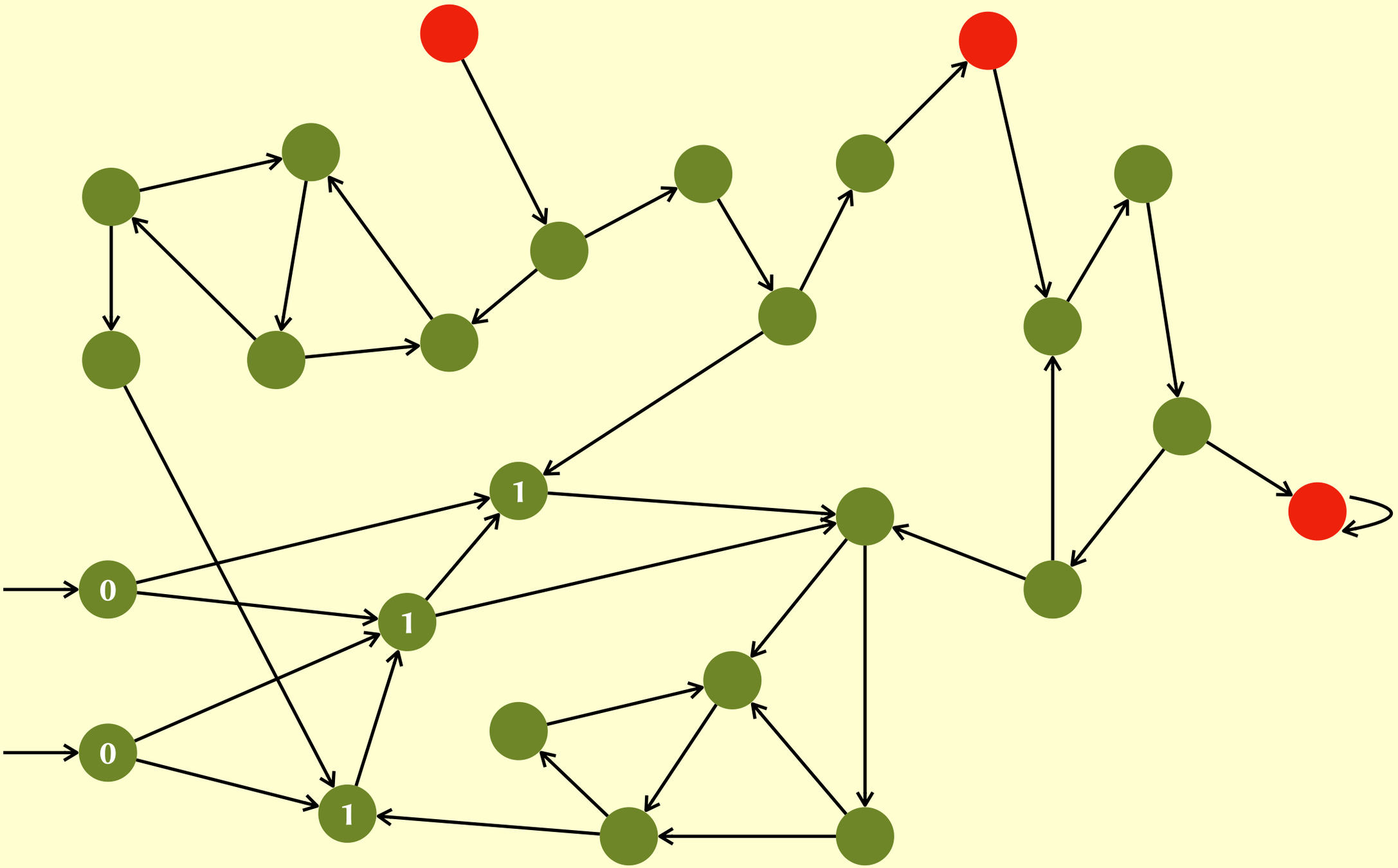
Bounded model checking



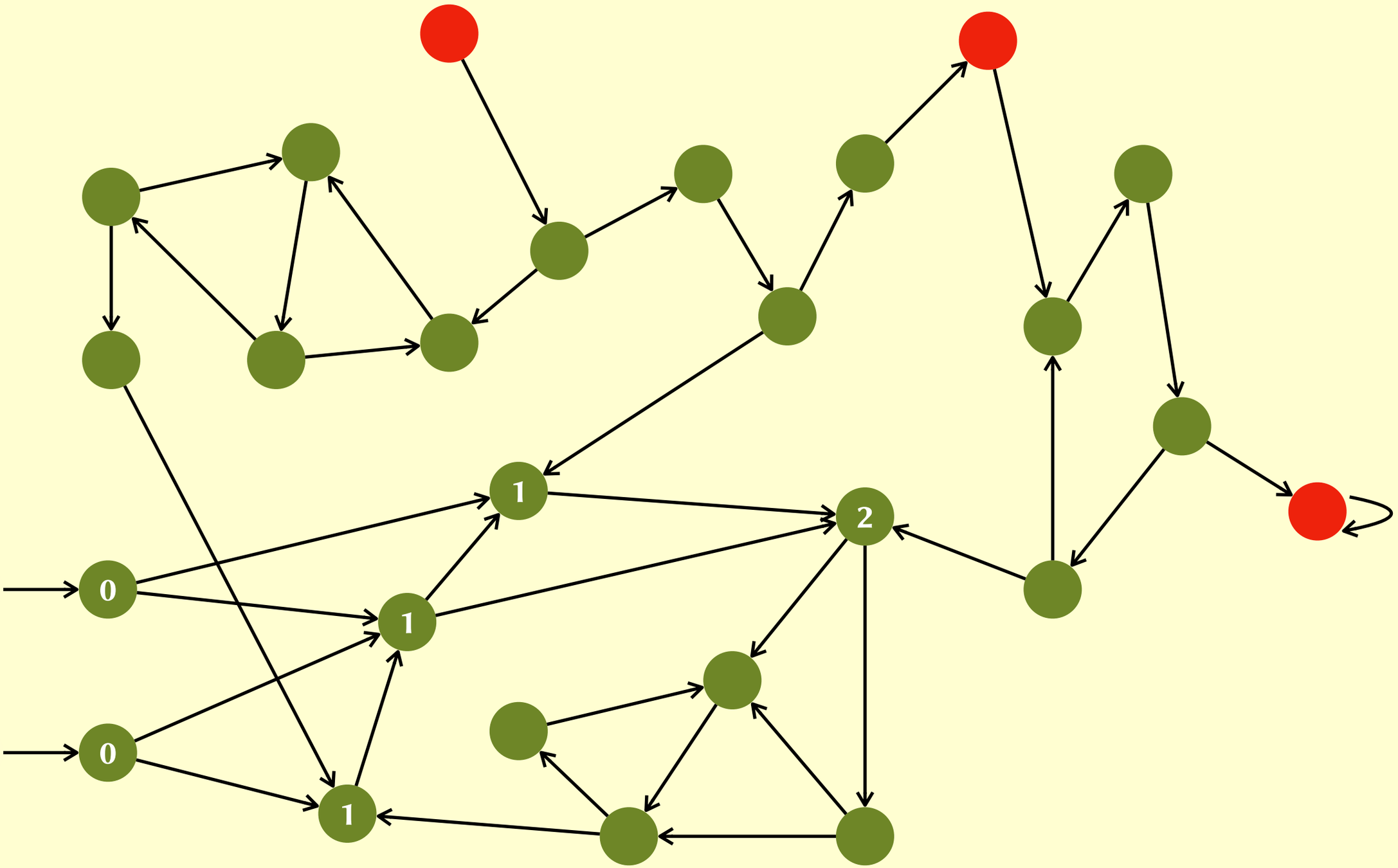
Bounded model checking



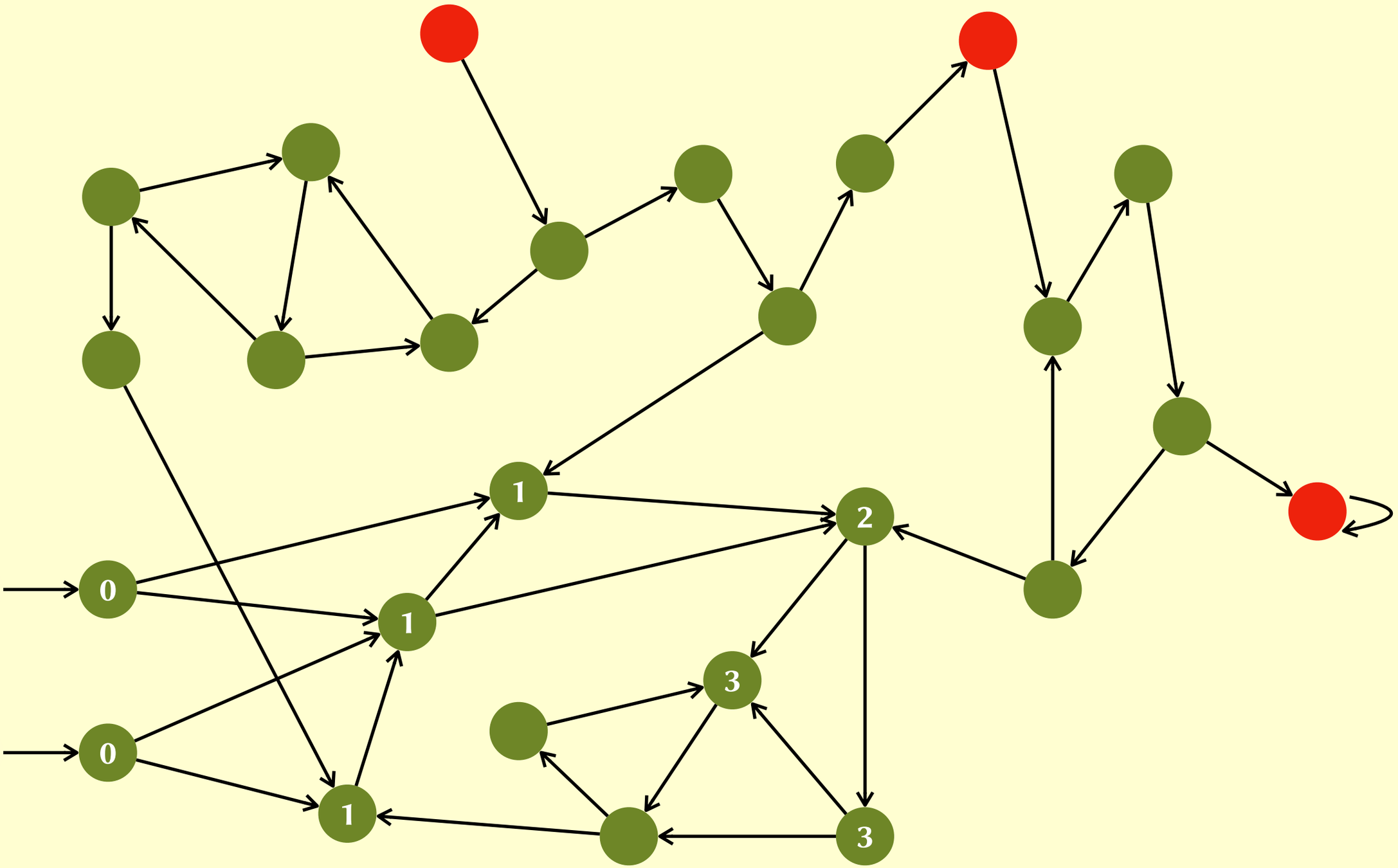
Bounded model checking



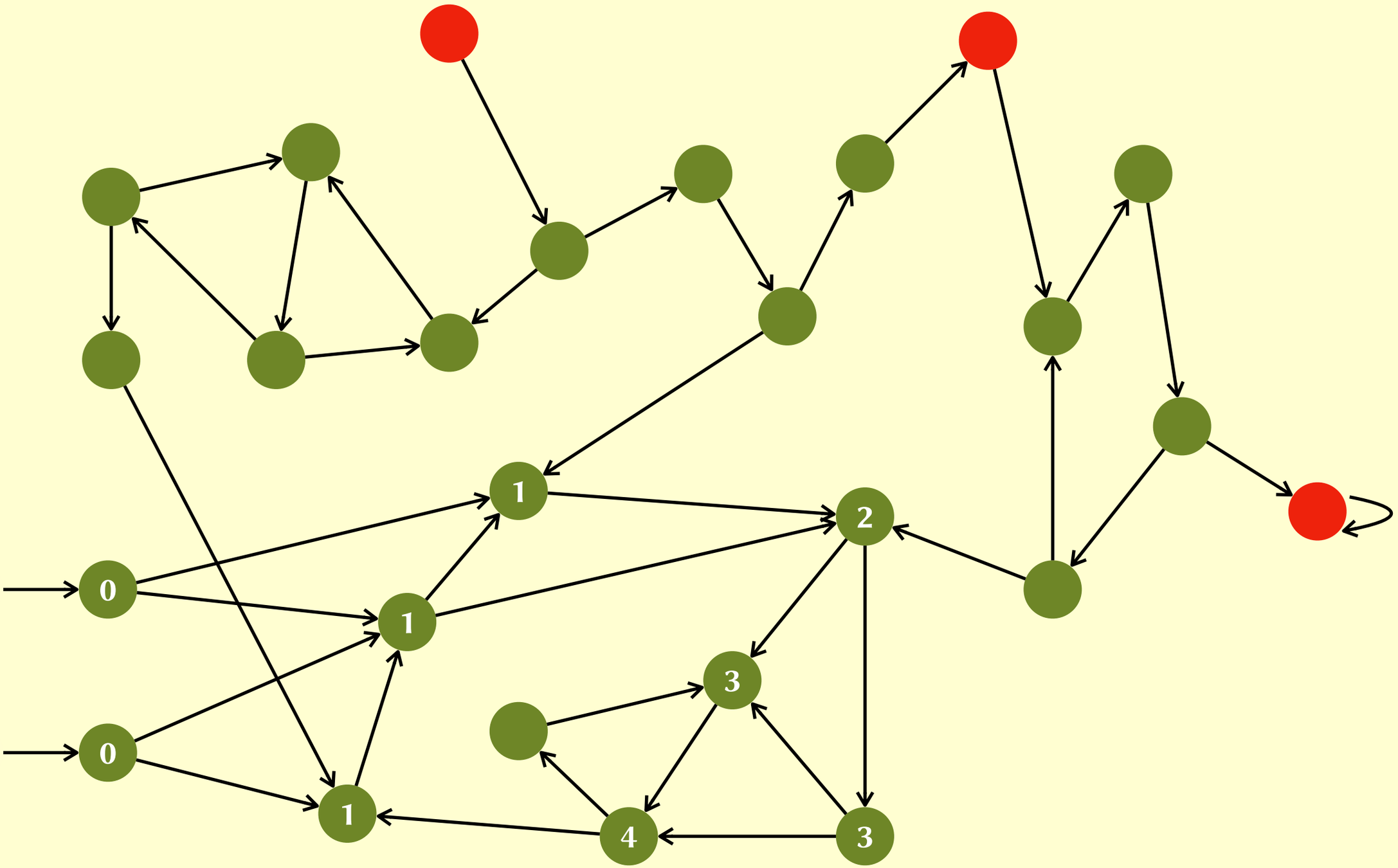
Bounded model checking



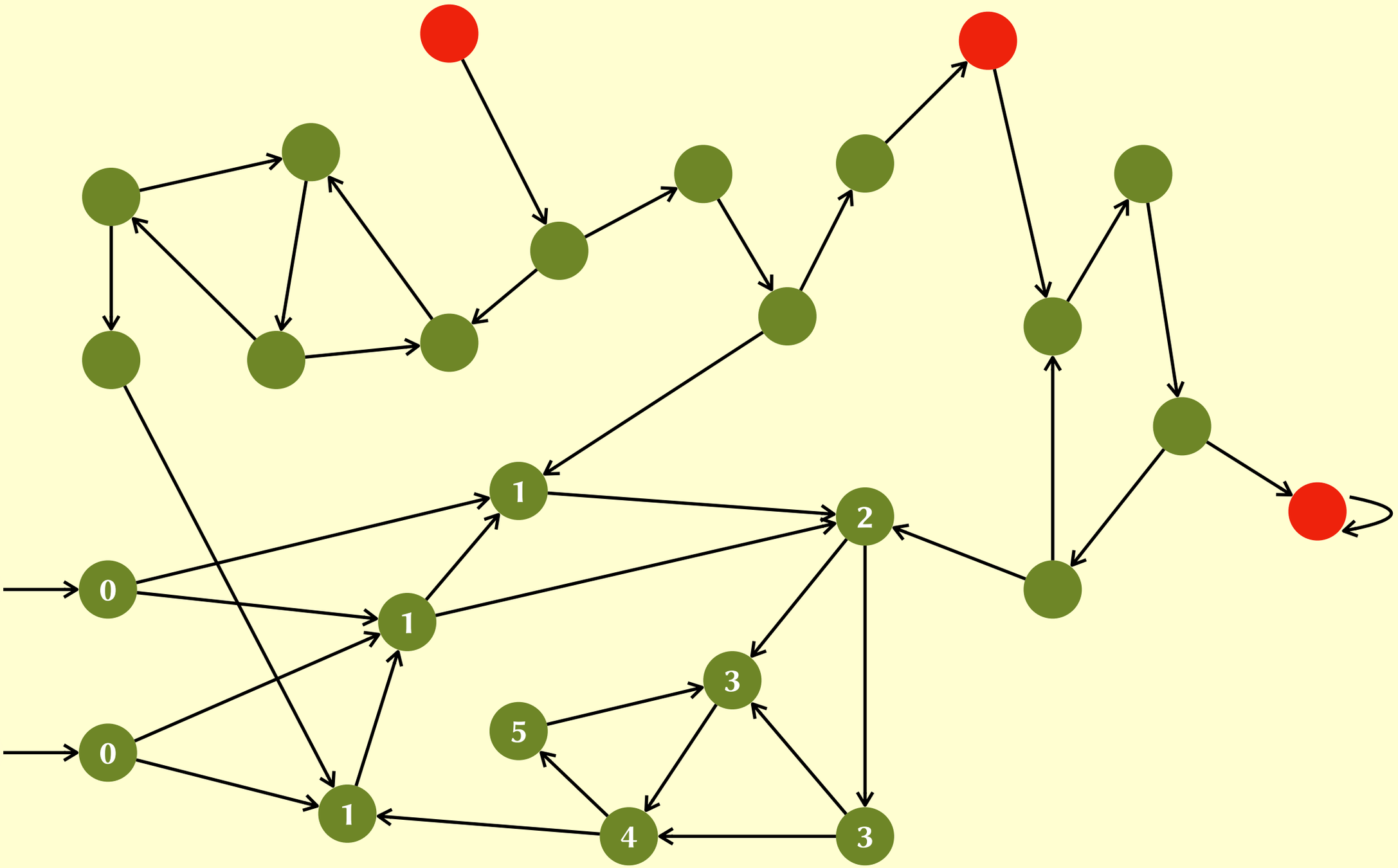
Bounded model checking



Bounded model checking



Bounded model checking



Bounded model checking

- **Simulation** considers only a *single* path at a time, and can only follow it for a *bounded* number of steps.
  - Guarantee provided by N-cycle simulation: if you provide *these specific inputs*, nothing goes wrong within N cycles.
- **BMC** considers *all* paths simultaneously, but can still only follow them for a bounded number of steps.
  - Guarantee provided by BMC with bound N: with *any inputs*, nothing goes wrong within N cycles.

SymbiYosys

- SymbiYosys is part of the Yosys suite of open-source hardware synthesis tools.

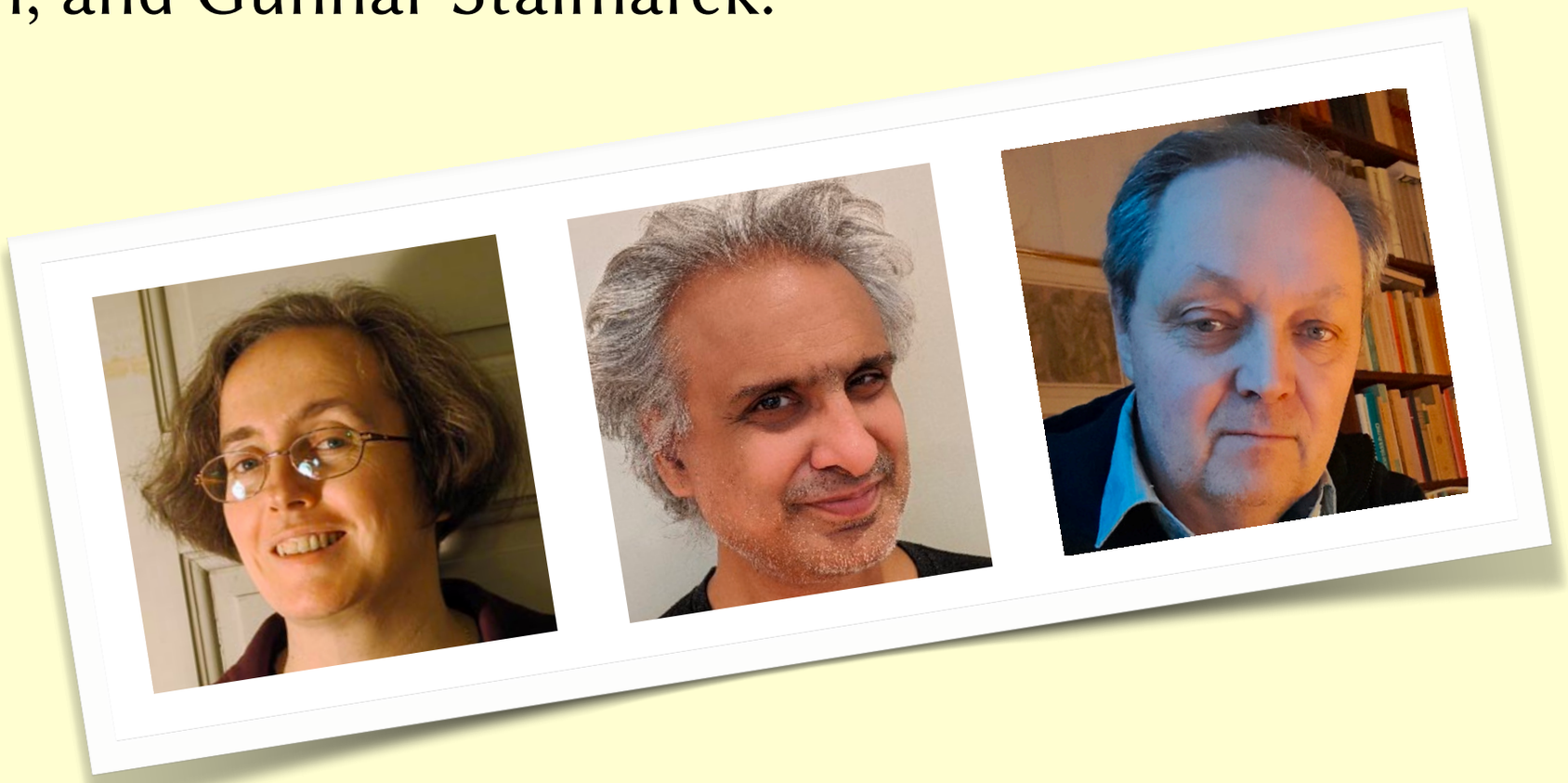

Claire Xenia Wolf

- Lead developer is Claire Xenia Wolf, since 2012.

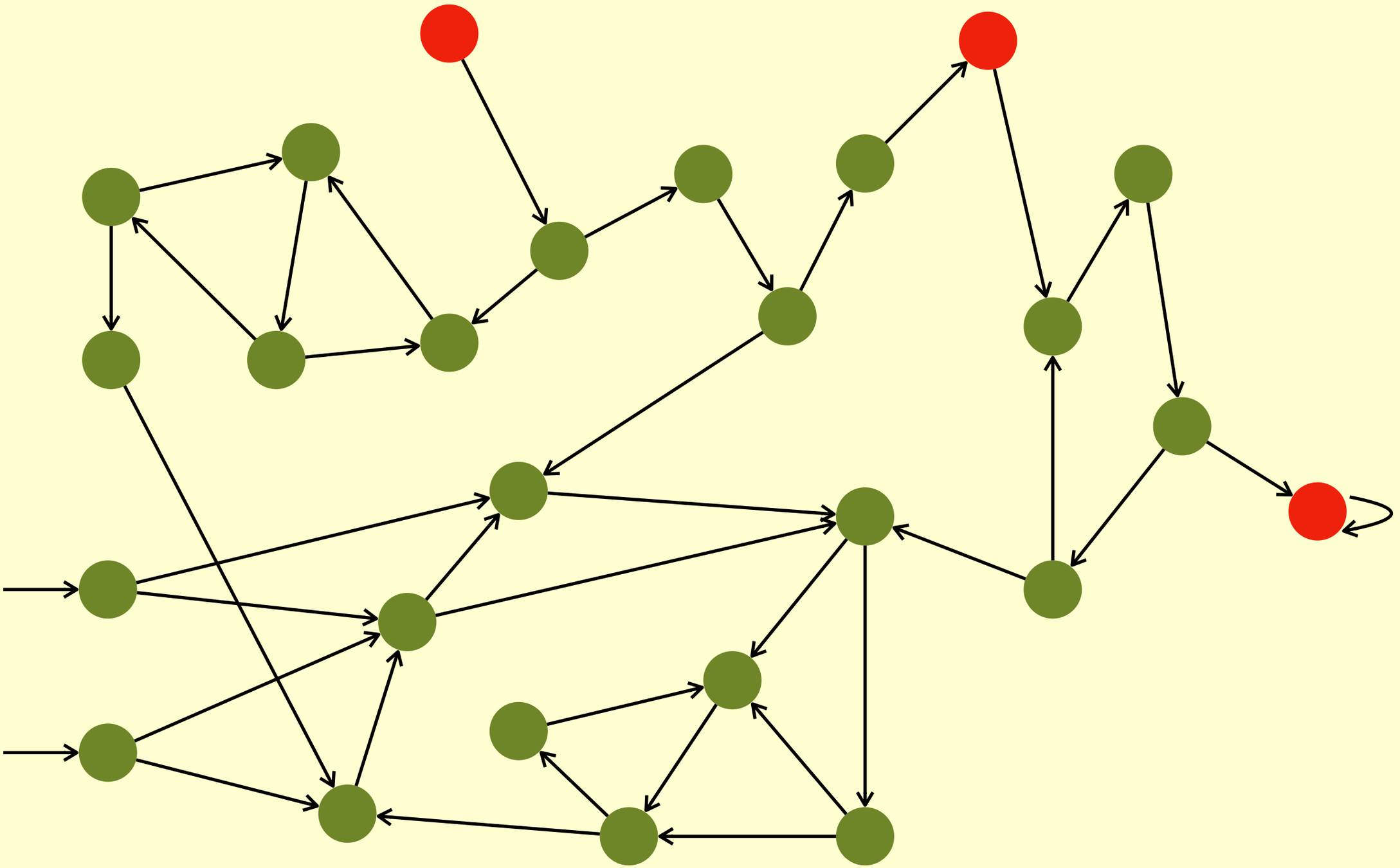
- Installation instructions are at: `https://symbiyosys.readthedocs.io/`

k-induction

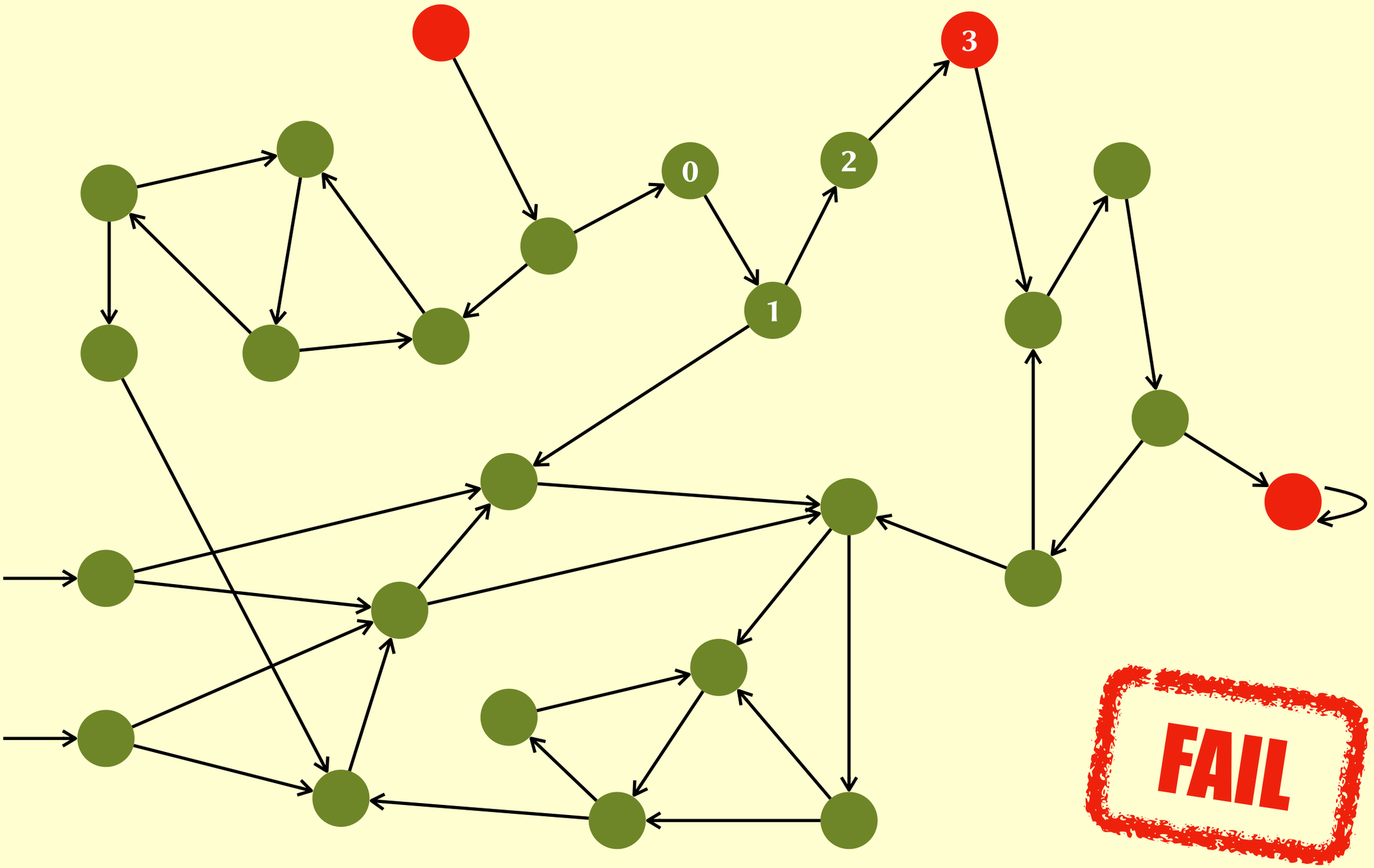
- Invented around 2000 by Mary Sheeran, Satnam Singh, and Gunnar Stålmarck.



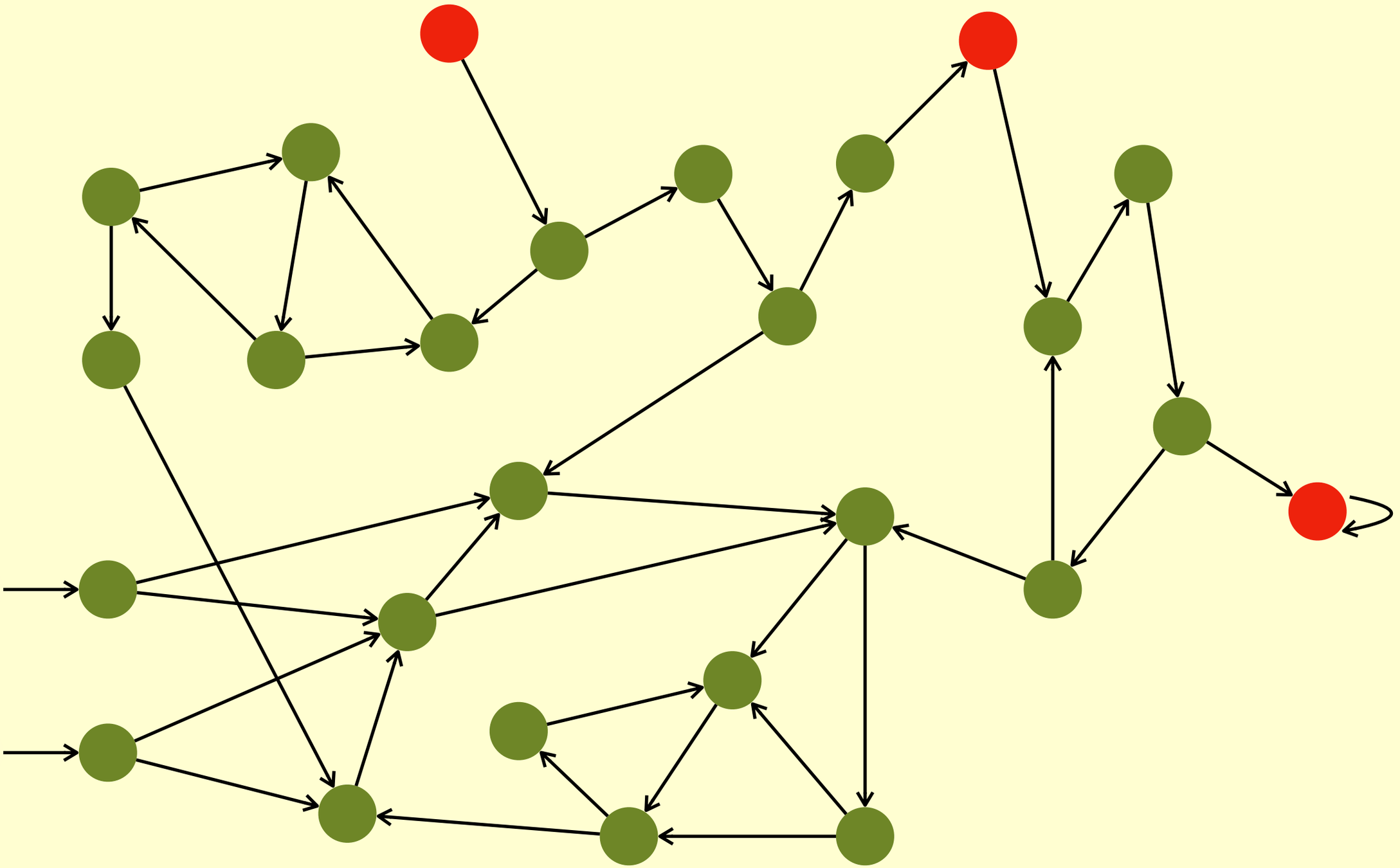
k-induction (k=3)



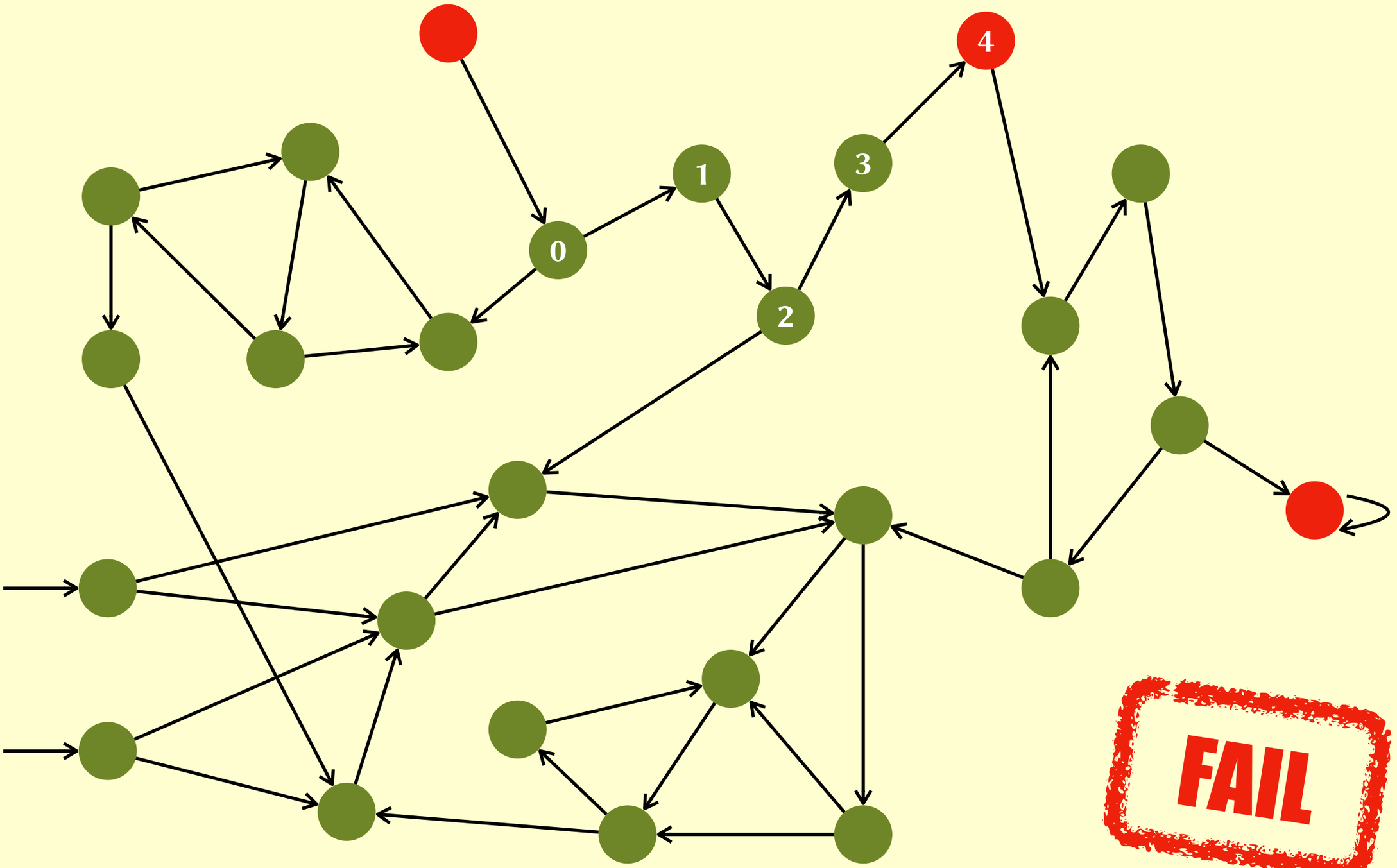
k-induction (k=3)



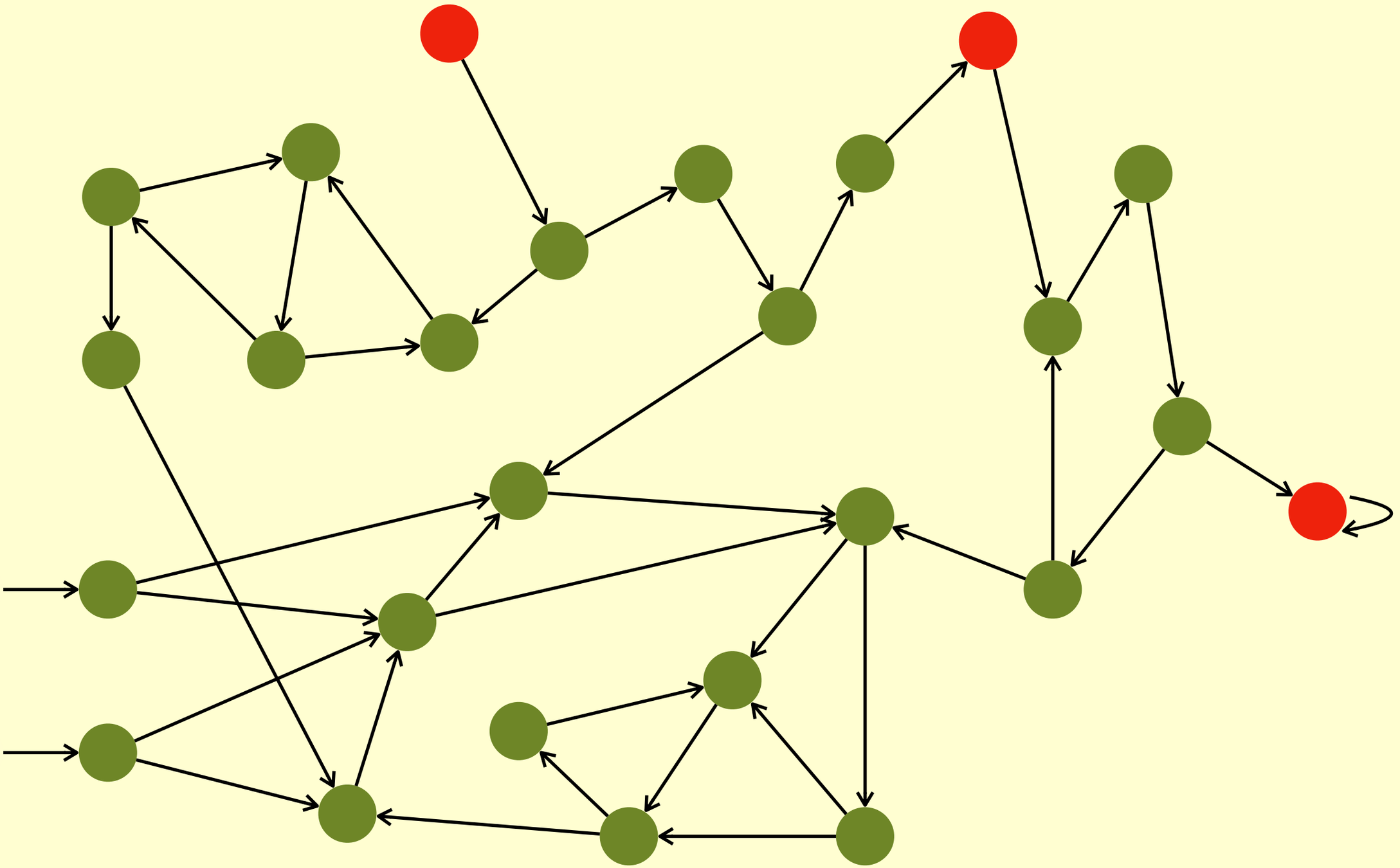
k-induction (k=4)



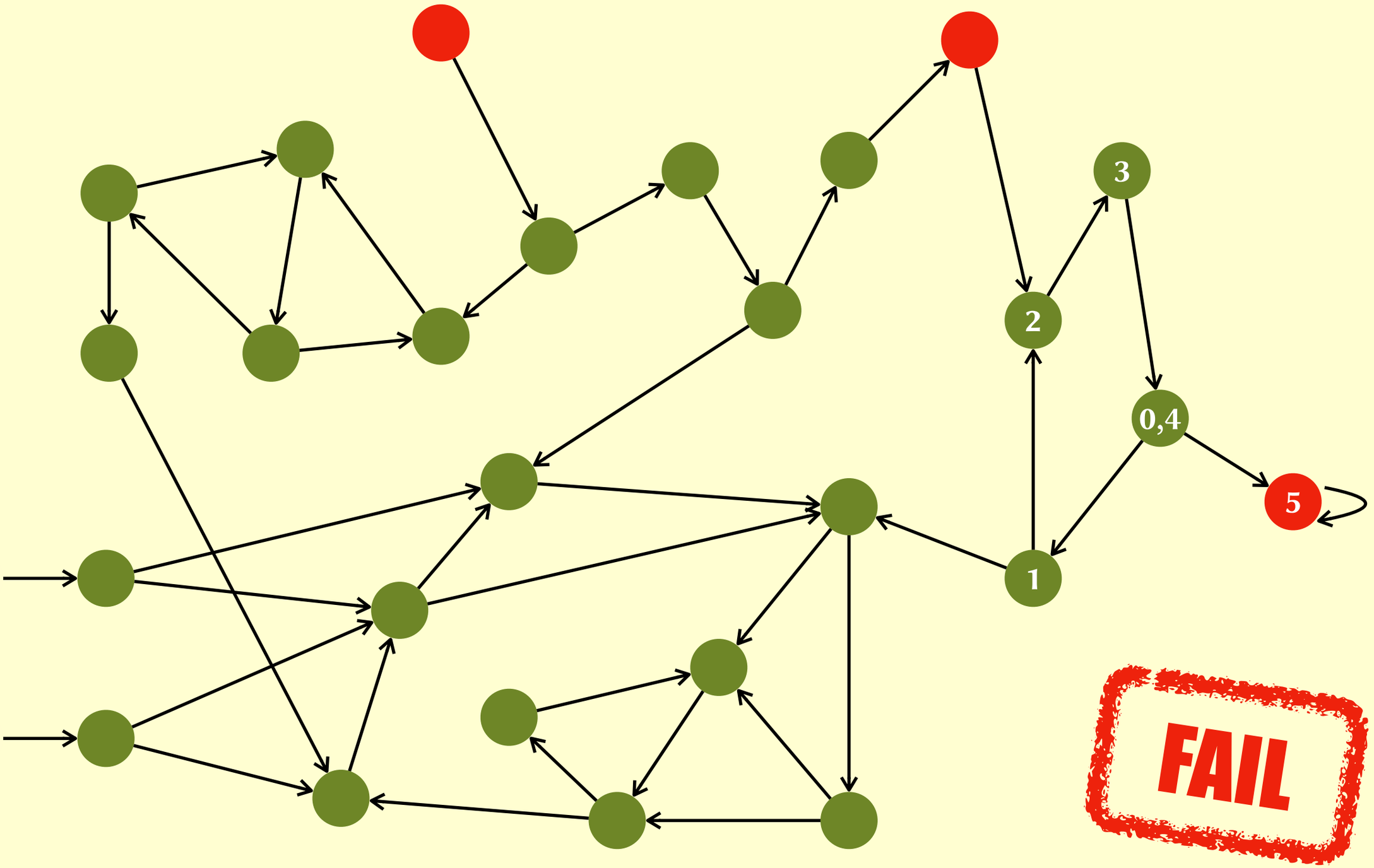
k-induction (k=4)



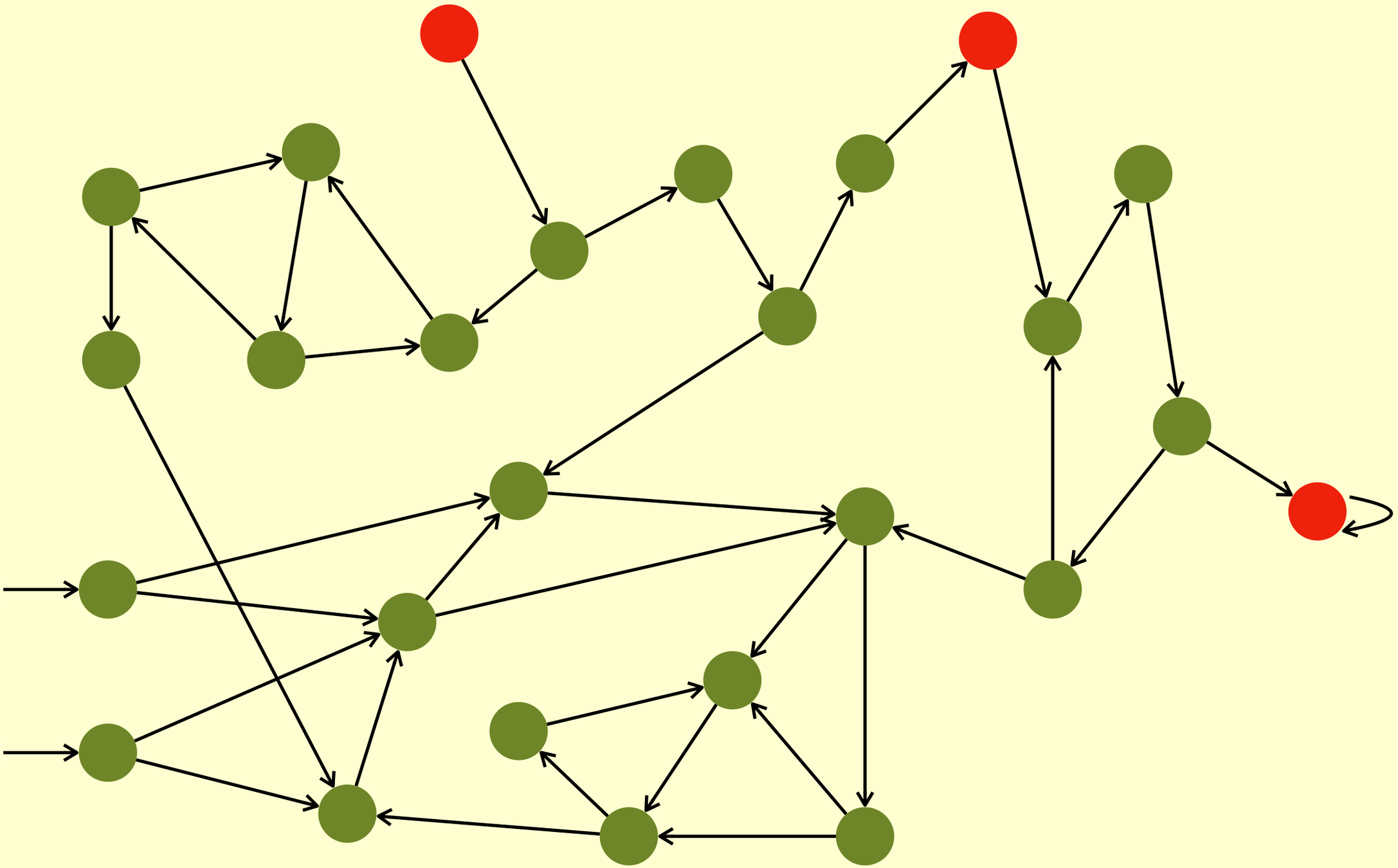
k-induction (k=5)



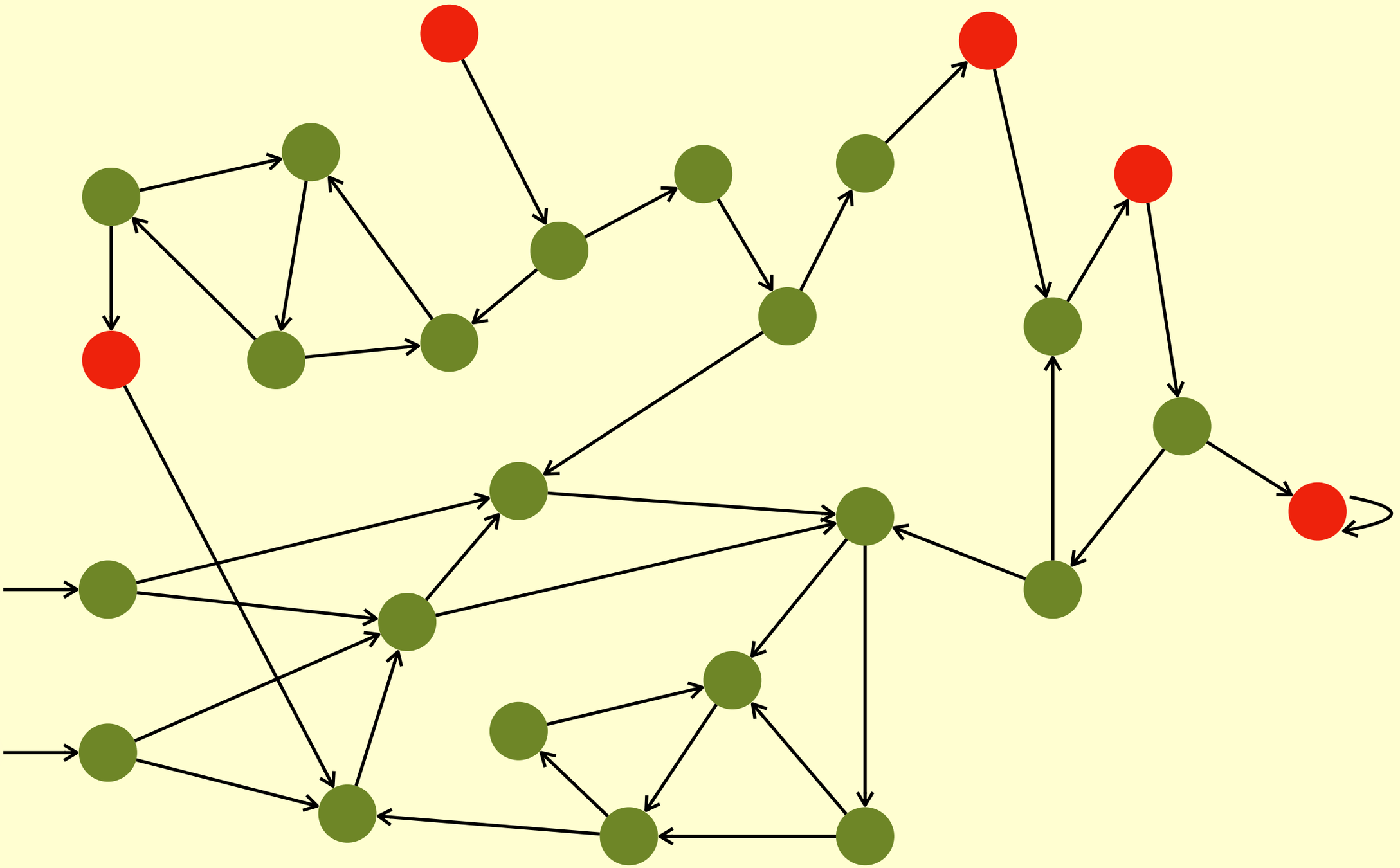
k-induction (k=5)



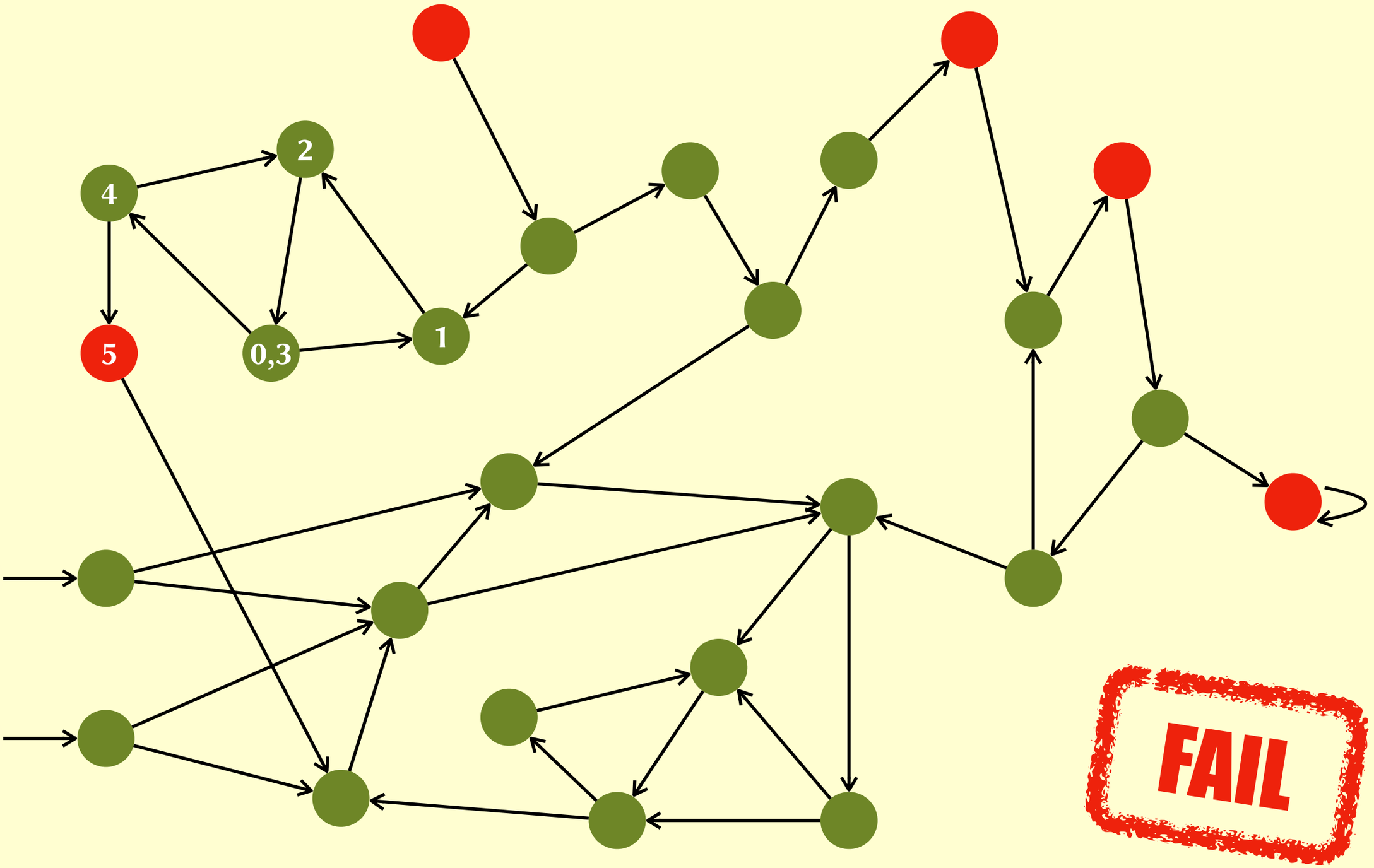
Need more assertions!



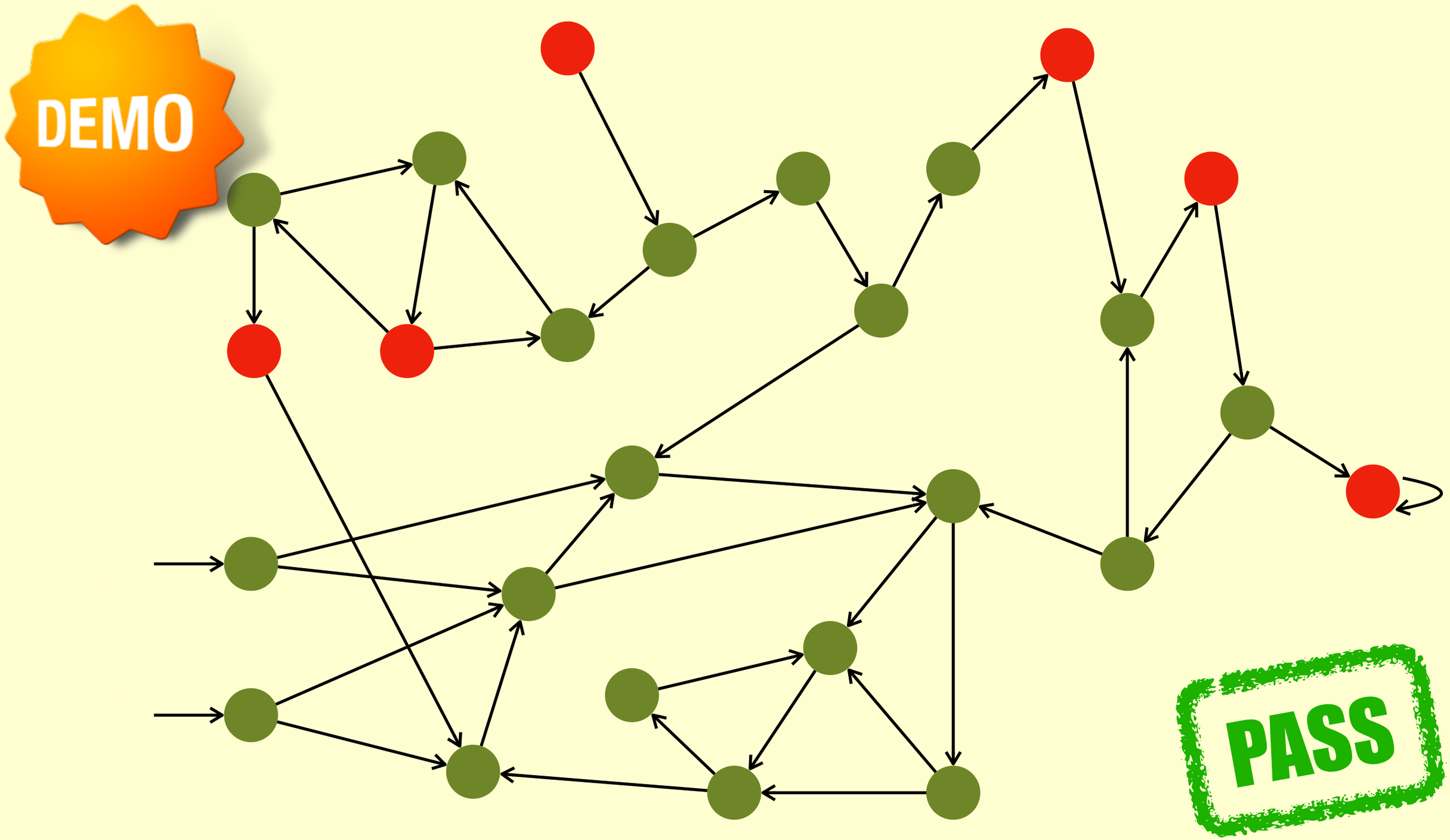
Need more assertions!



k-induction (k=5)



Even more assertions!



BMC vs k-induction

- BMC (with bound N) proves:
  for every k<N
  and for any chain of states $P_0 \longrightarrow P_1 \longrightarrow \ldots \longrightarrow P_k$,
  if P_0 is an initial state
  then P_k is a good state.

- k-induction proves:
  for any chain of states $P_0 \longrightarrow P_1 \longrightarrow \ldots \longrightarrow P_k$,
  if P_0, ..., P_{k-1} are all good states
  then P_k is also a good state.

BMC vs k-induction

- BMC asks a SAT solver to solve formulas of the form:

$$\text{init}(P_0) \wedge \text{next}(P_0,P_1) \wedge \ldots \wedge \text{next}(P_{k-1},P_k) \wedge \text{bad}(P_k)$$

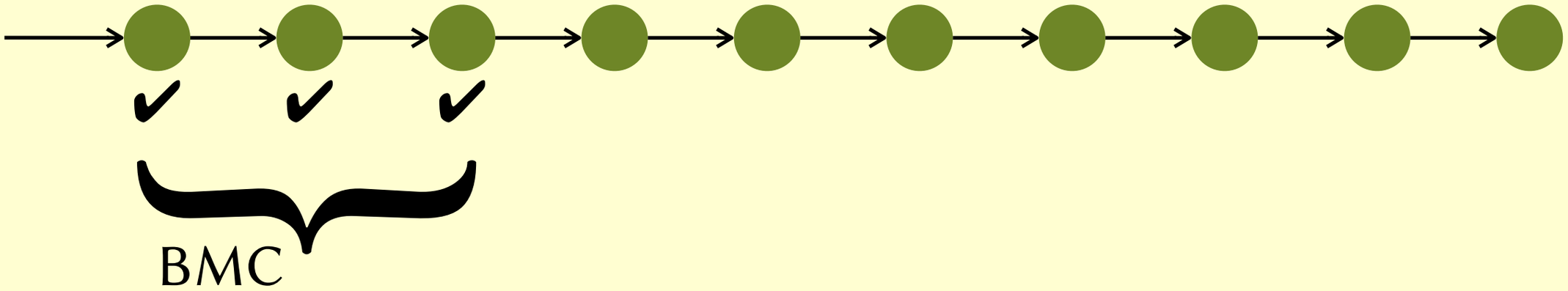
- k-induction asks a SAT solver to solve formulas of the form:

$$\text{next}(P_0,P_1) \wedge \ldots \wedge \text{next}(P_{k-1},P_k) \wedge \text{good}(P_0) \wedge \ldots \wedge \text{good}(P_{k-1}) \wedge \text{bad}(P_k)$$

- In both cases, the proof is complete if the solver fails!

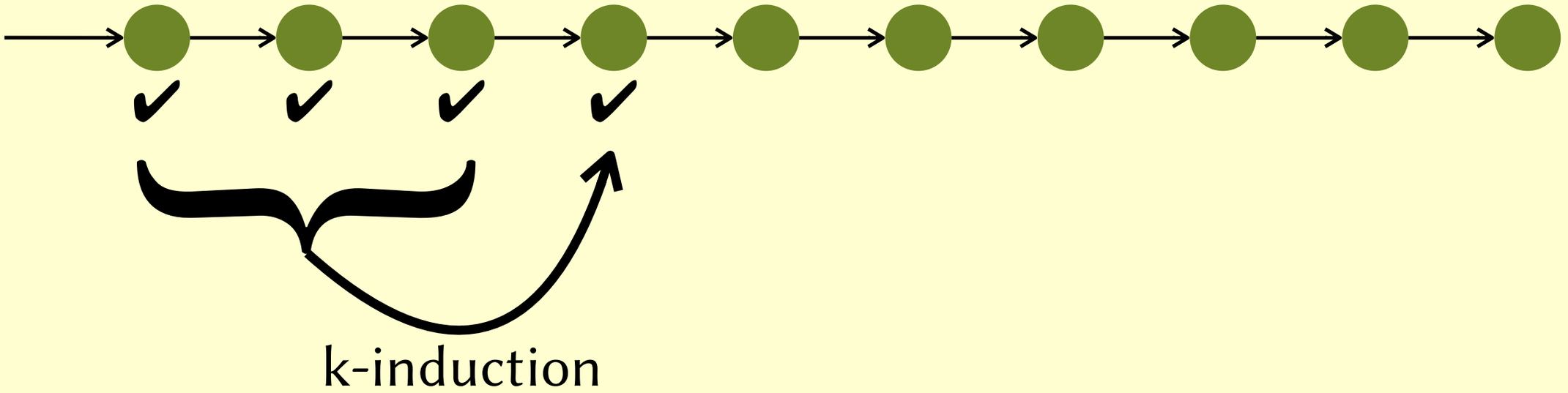
BMC vs k-induction

- The two techniques combine powerfully.



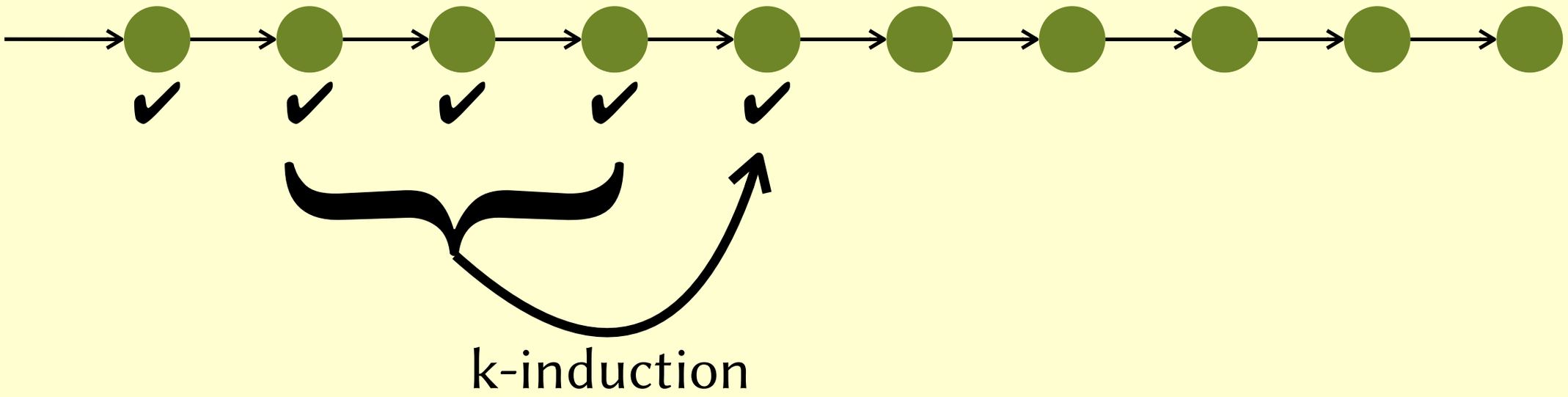
BMC vs k-induction

- The two techniques combine powerfully.



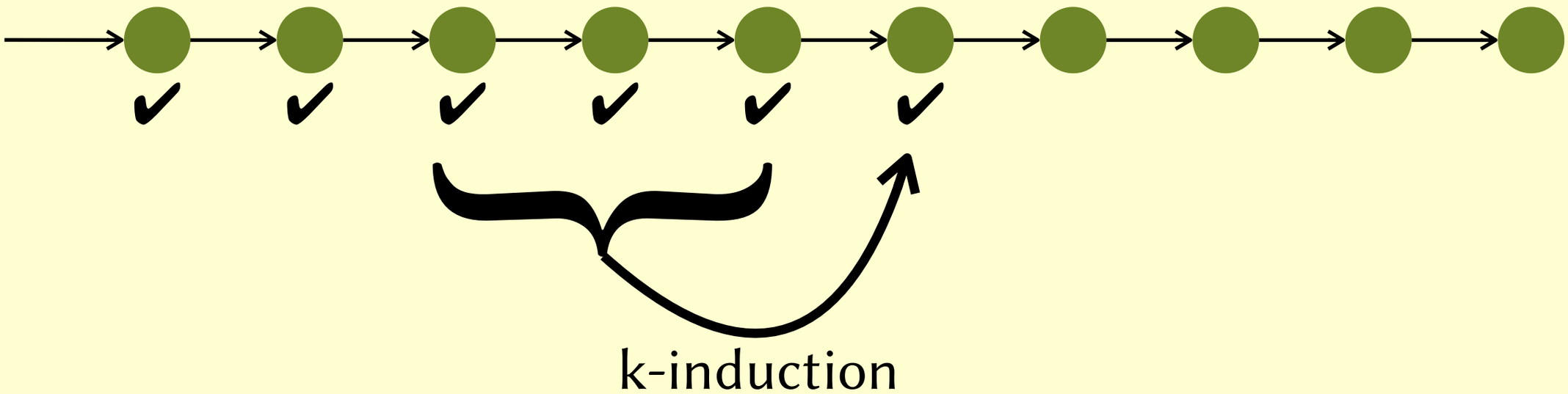
BMC vs k-induction

- The two techniques combine powerfully.



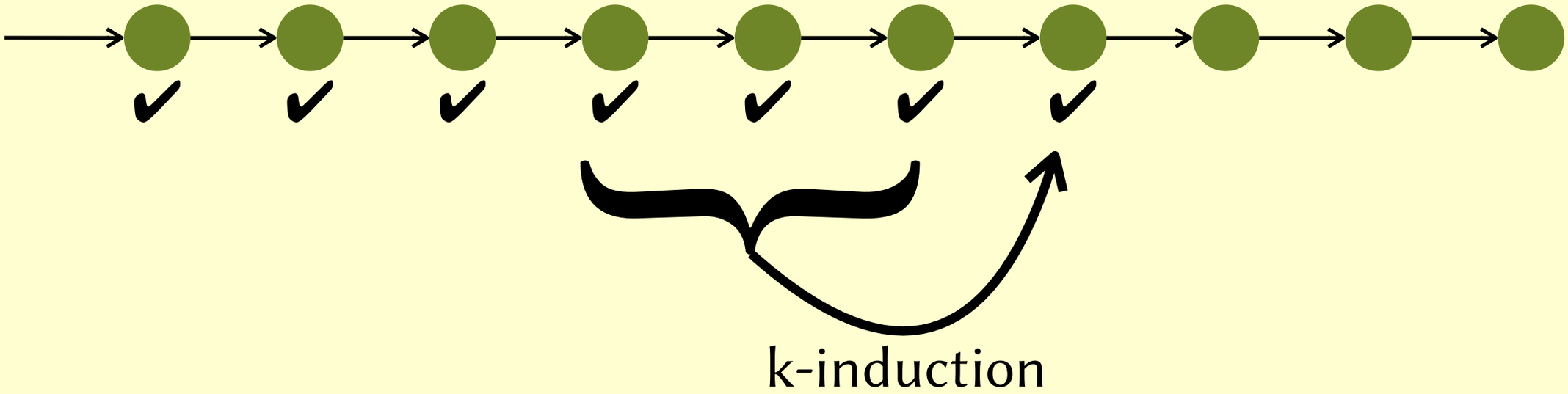
BMC vs k-induction

- The two techniques combine powerfully.



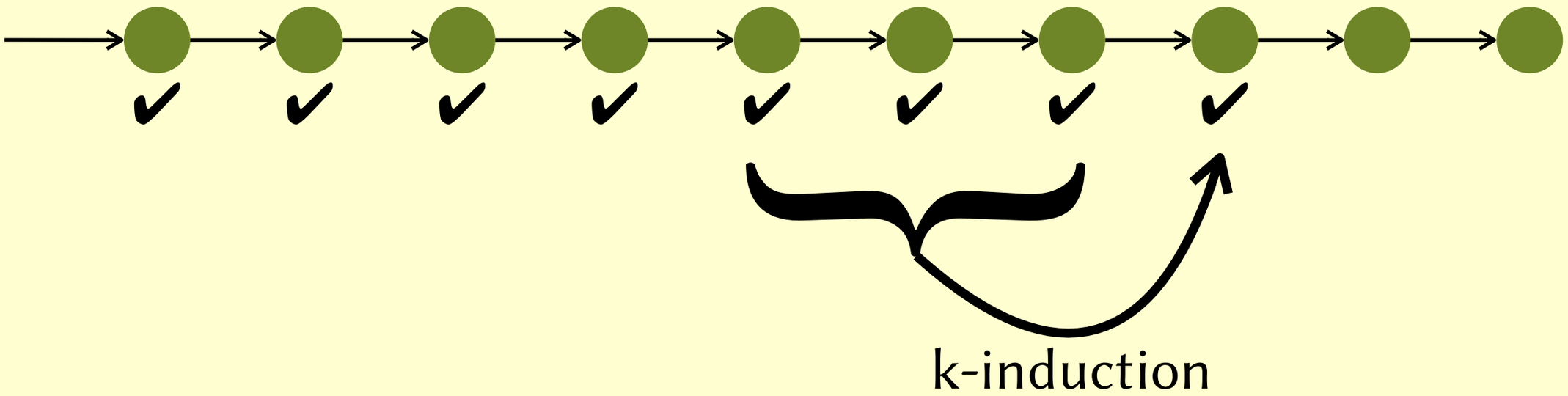
BMC vs k-induction

- The two techniques combine powerfully.



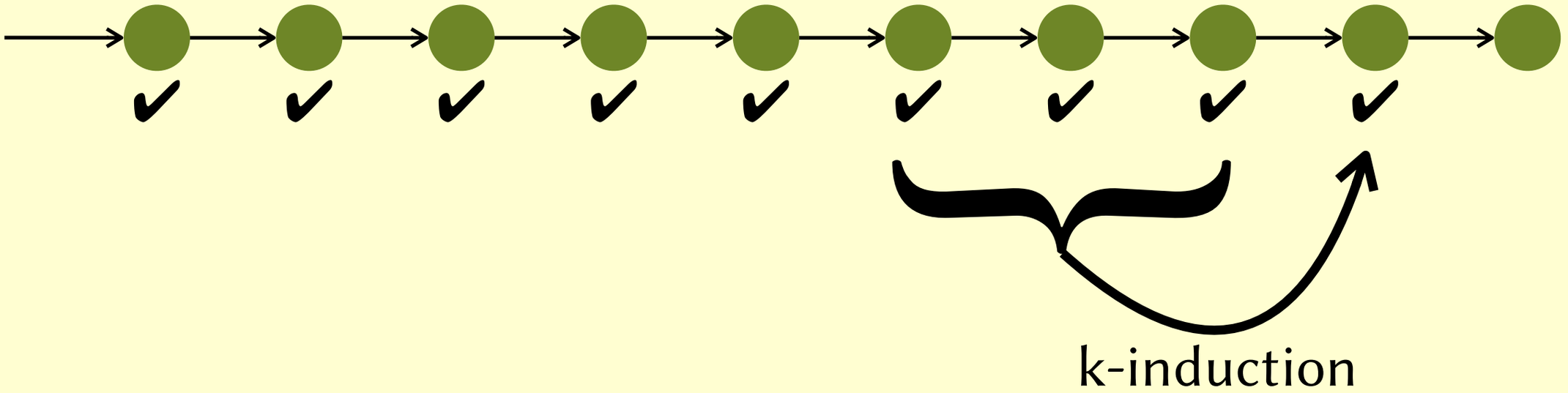
BMC vs k-induction

- The two techniques combine powerfully.



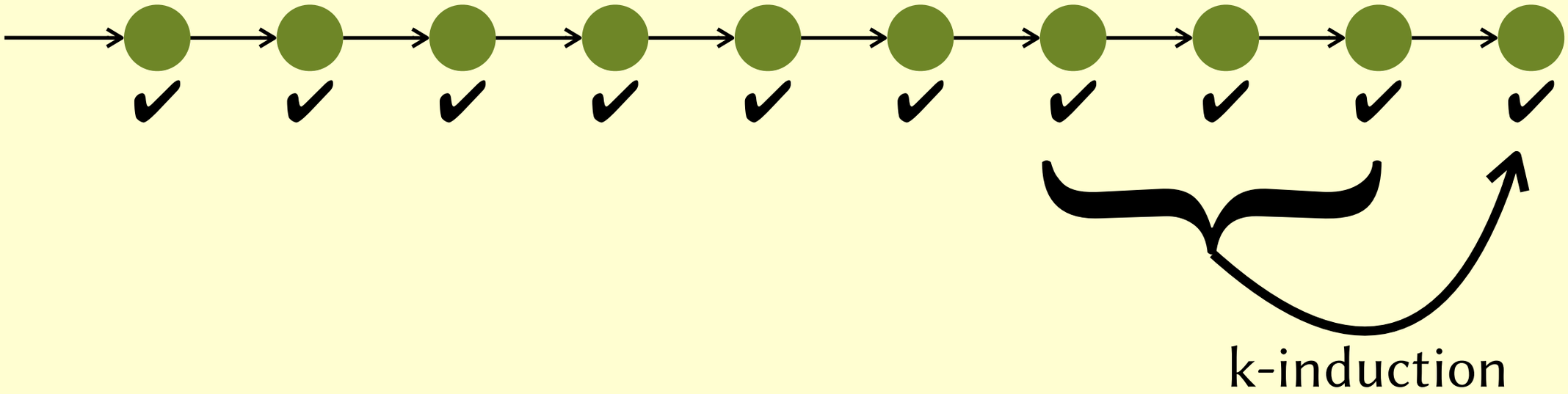
BMC vs k-induction

- The two techniques combine powerfully.



BMC vs k-induction

- The two techniques combine powerfully.



Simulation vs BMC vs k-induction

- Guarantee provided by N-cycle **simulation**:
  if you provide *these specific inputs*,
  nothing goes wrong within N cycles.

- Guarantee provided by **BMC** with bound N:
  with *any inputs*,
  nothing goes wrong within N cycles.

- Guarantee provided by **k-induction**:
  with *any inputs*,
  nothing goes wrong after *any number* of cycles.