Algorithms for LTL model checking

Existential model checking: $[\![\mathcal{K}]\!] \cap \mathcal{B}_\phi \neq \emptyset$

Universal model checking: $[\![\mathcal{K}]\!] \cap \mathcal{B}_{\neg\phi} = \emptyset$

Typical instances:

- Size of $\mathcal{K}$: between several hundreds to millions of states.
- Size of $\mathcal{B}_{\neg\phi}$: exponential in $|\phi|$, but comparatively small.

Typical setting:

- $\mathcal{K}$ indirectly given by some concise description (modelling or programming language); model-checking tools will generate $\mathcal{K}$ internally.
- $\mathcal{B}_{\neg\phi}$ can be generated from ϕ before start of emptiness check.
- Example: SPIN model-checking tool

On-the-fly model-checking for LTL

$\mathcal{B}$ generated "on-the-fly" from (some description of) $\mathcal{K}$ and from $\mathcal{B}_{\neg\phi}$ and tested for emptiness *at the same time*.

Size of $\mathcal{B}$ not known initially!

At the start, only the initial state is known, and some function $\mathtt{succ}\colon S \to 2^S$ computes immediate successors of a given state

Can stop exploration when counterexample found.

Strongly connected components

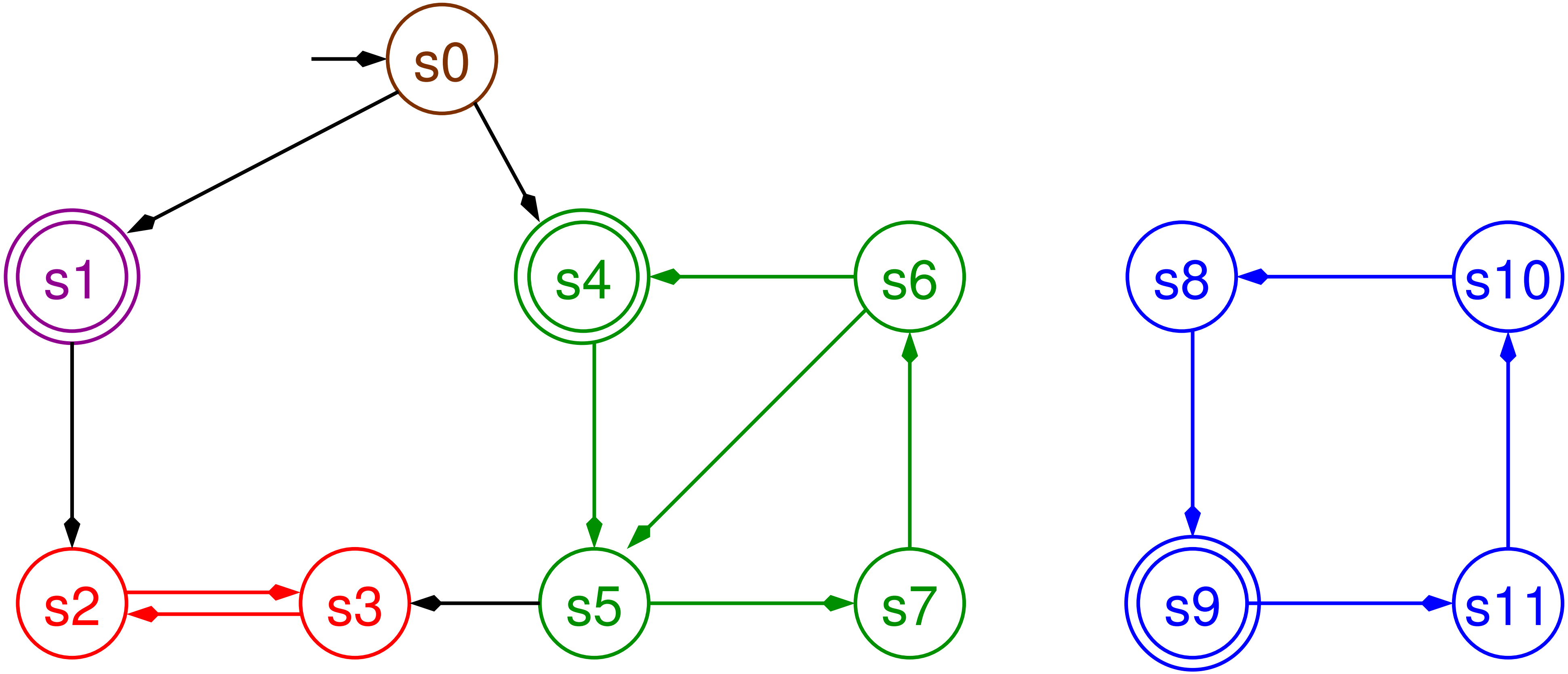
Let S be the set of states in $\mathcal{B}$.

$C \subseteq S$ is called a strongly connected component (SCC) iff

- $s \rightarrow^* s'$ for all $s, s' \in C$;
- C is maximal w.r.t. the above property, i.e. there is no proper superset of C satisfying the above.

An SCC C is called trivial if $|C| = 1$ and for the unique state $s \in C$ we have $s \not\rightarrow s$ (single state without loop).

Example: SCCs



The SCCs $\{s_0\}$ and $\{s_1\}$ are trivial.

SCCs and model-checking

Fact: $\mathcal{B}$ contains a counterexample iff it contains a reachable non-trivial SCC with an accepting state.

Most on-the-fly MC algorithms are based on depth-first search:

Tarjan's SCC algorithm

Nested DFS (used by Spin)

Improved SCC detection (Couvreur et al)

Depth-first search (basic version)

```
nr = 0;
hash = {};
dfs(s0);
exit;

dfs(s) {
   add s to hash;
   nr = nr+1;
   s.num = nr;

   for (t in succ(s)) {
      // deal with transition s -> t
      if (t not yet in hash) { dfs(t); }
   }
}
```

Solution (1): Tarjan's algorithm

The algorithm of Tarjan (1972) can identify the SCCs in linear time (i.e. proportional to $|S| + |\delta|$).

Said algorithm is an extension of basic DFS with additional constant-time operations on each state and transition.

When identifying an SCC, check if it is non-trivial and contains accepting state.

Memory usage: (mostly) one integer per state

Solution (2): nested DFS

Algorithm proposed by Courcoubetis, Vardi, Wolper, Yannakakis (1992).

The nested-DFS algorithm is an alternative requiring only two bits per state.

States are "white" initially.

A first DFS makes all the states that it visits blue.

Whenever the first (blue) DFS backtracks from an *accepting* state s, it starts a second (red) DFS to see if there is a cycle around s.

The red DFS only visits states that are not already red (e.g. from a previous red DFS). Thus, every state and edge is considered at most twice.

Nested depth-first search: Algorithm

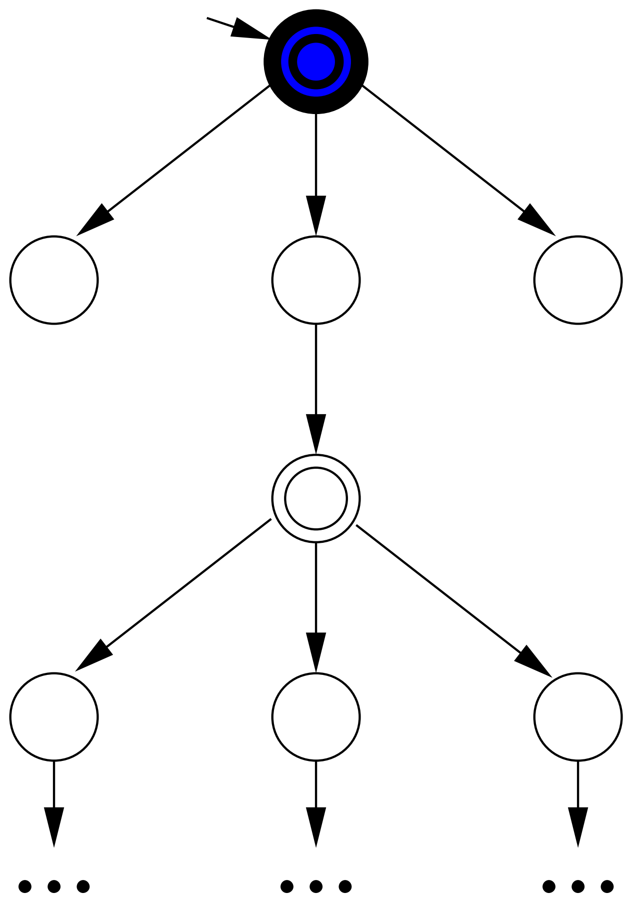
```
hash = {};
blue(s0);
report "no accepting run"

blue(s) {
   add (s,0) to hash;
   for t in succ(s)
      if (t,0) not in hash { blue(t) }
   if s is accepting and (s,1) not in hash { seed=s; red(s) }
}

red(s) {
   add (s,1) to hash;
   for t in succ(s)
      if t=seed { report "accepting run found"; exit }
      if (t,1) not in hash { red(t) }
}
```

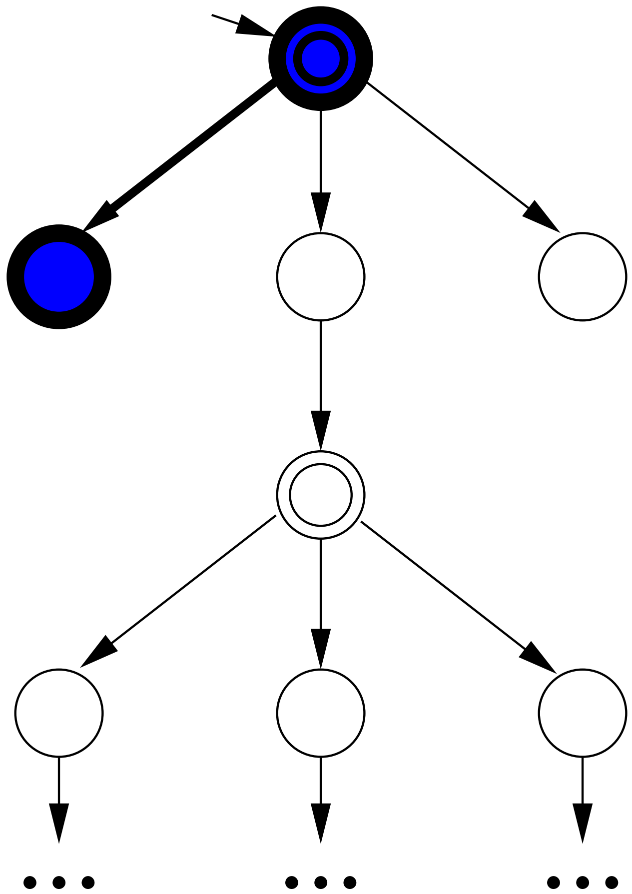
Nested DFS: Example

Blue phase: Start at initial state.



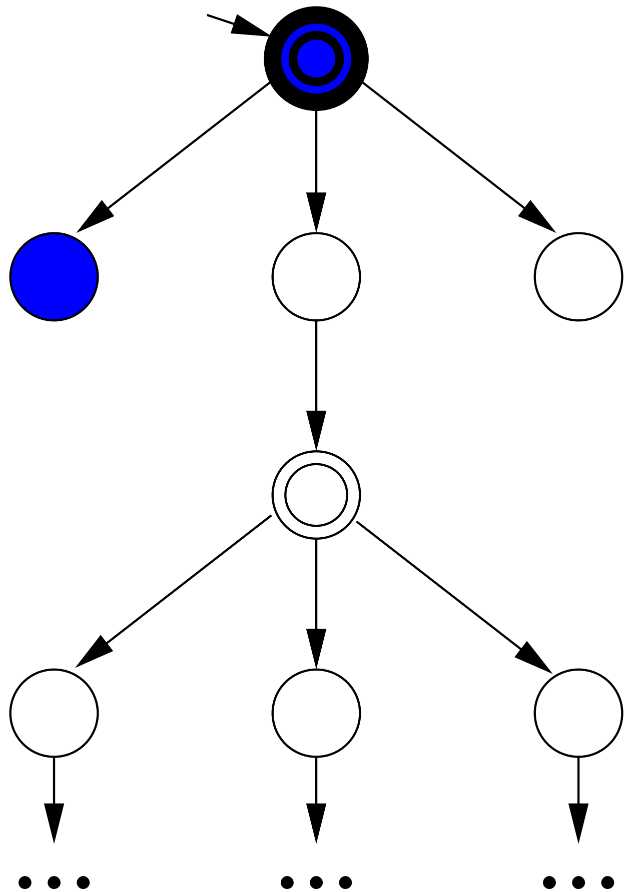
Nested DFS: Example

Visit states depth-first, colouring them blue.



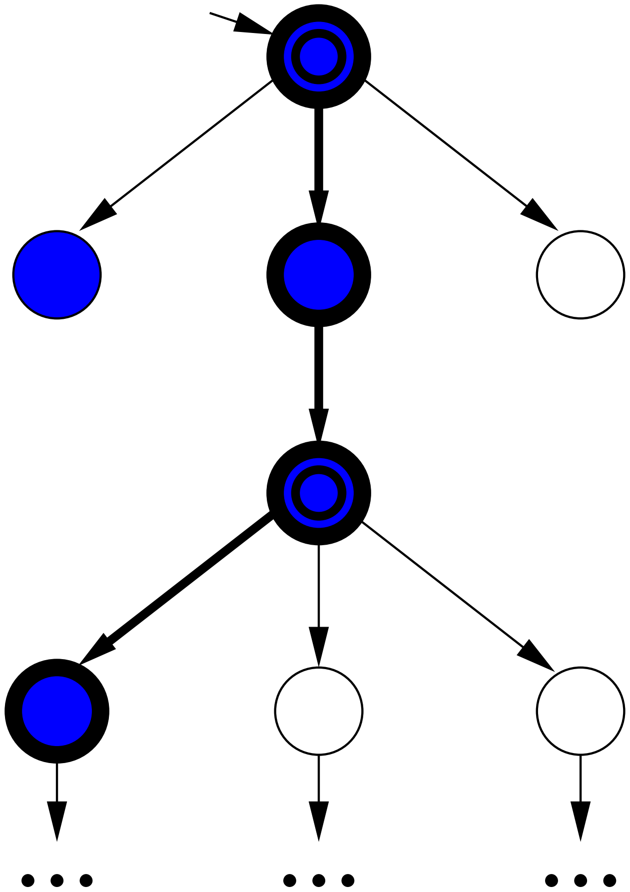
Nested DFS: Example

Simply backtrack from non-accepting states.



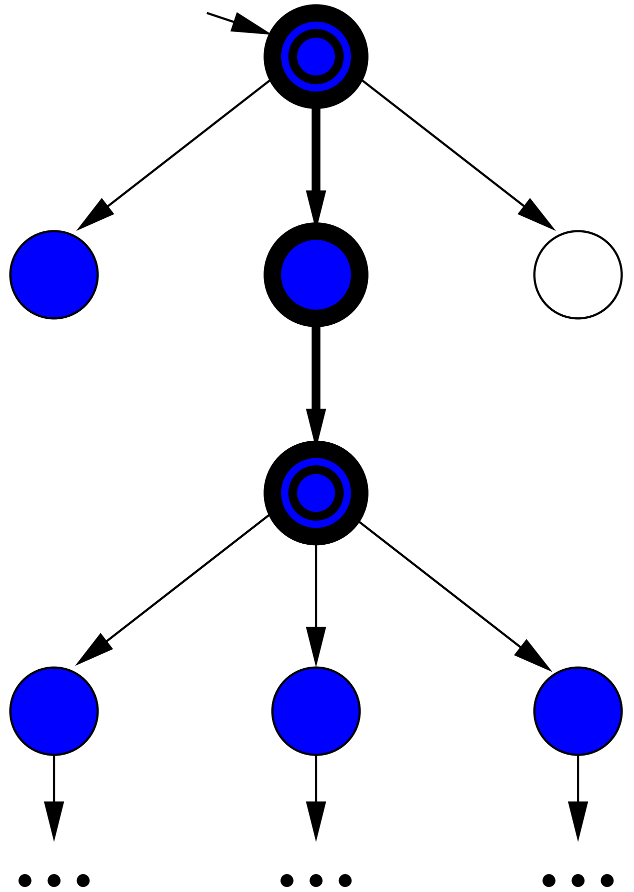
Nested DFS: Example

Continue blue search . . .



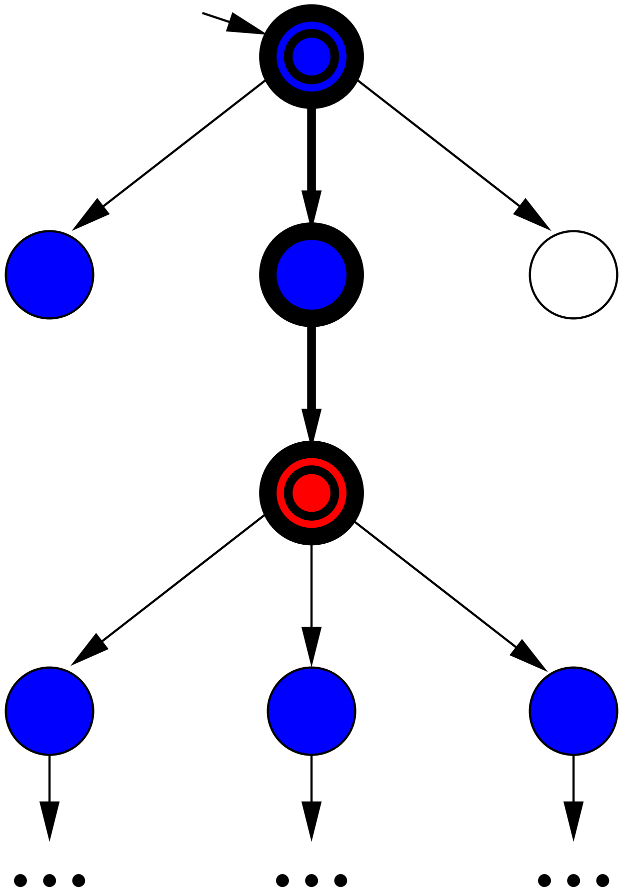
Nested DFS: Example

Continue blue search until backtracking from an accepting state.



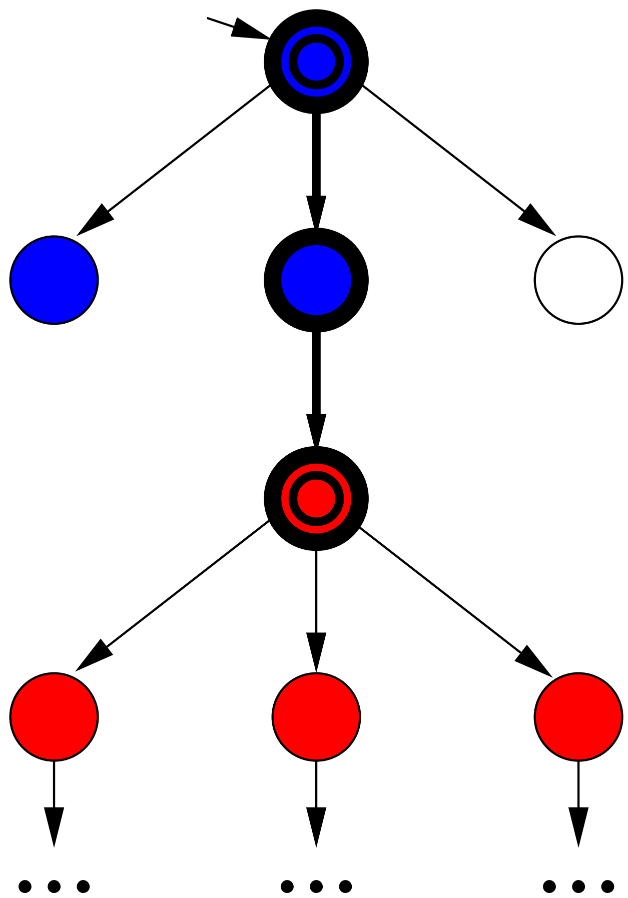
Nested DFS: Example

Before backtracking, start a “red” DFS . . .



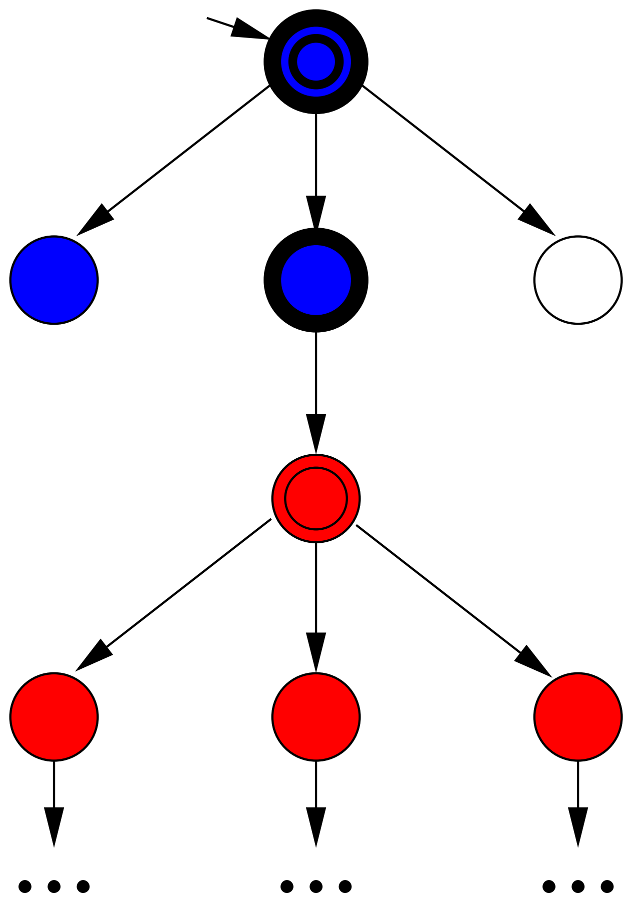
Nested DFS: Example

. . . that searches for a loop back to that accepting state.



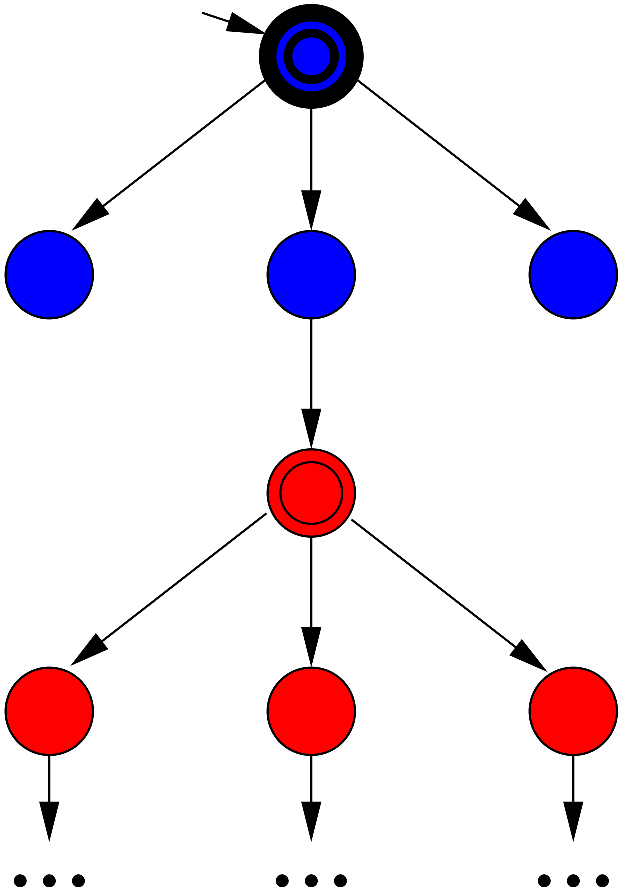
Nested DFS: Example

If red search is unsuccessful, backtrack.



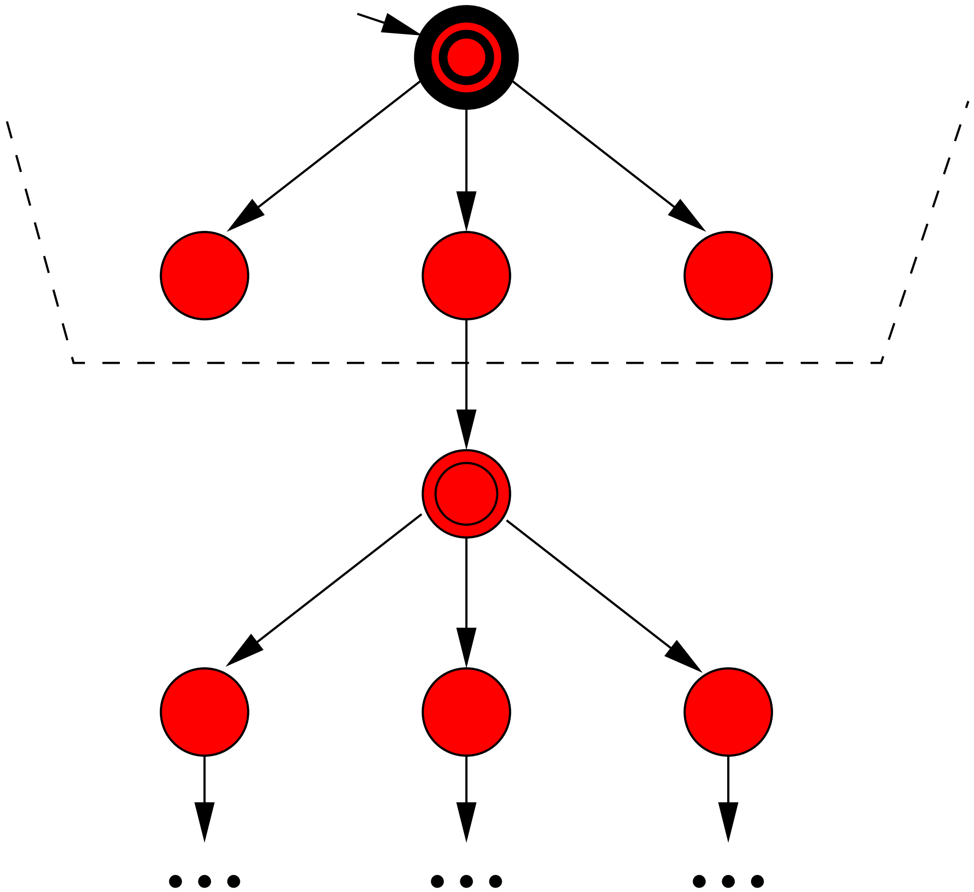
Nested DFS: Example

Carry on . . .



Nested DFS: Example

Future red searches only consider non-red states.



Correctness (sketch)

Invariant: When a red search terminates unsuccessfully, none of the red states forms part of an accepting run.

The first red search in a non-trivial SCC is bound to succeed.

Red search can only be unsuccessful if started from trivial SCC.

The first visited state of an SCC (its *root*) is also the last from which one backtracks.

Before backtracking from a root, one has backtracked from all other SCCs reachable from it. Therefore, those SCCs did not contain any accepting run and can safely be coloured red.

Properties of Nested DFS

(good) Very economic in terms of memory

(good) Can be combined with further optimization (partial-order reduction)

(bad) Tends to prefer long counterexamples “deep down” in the state graph

Implemented in state-of-the-art tools like Spin

$\rightarrow$ variants of Tarjan (not shown) can identify counterexamples more quickly, but are less economic on memory and more difficult to combine with other optimizations