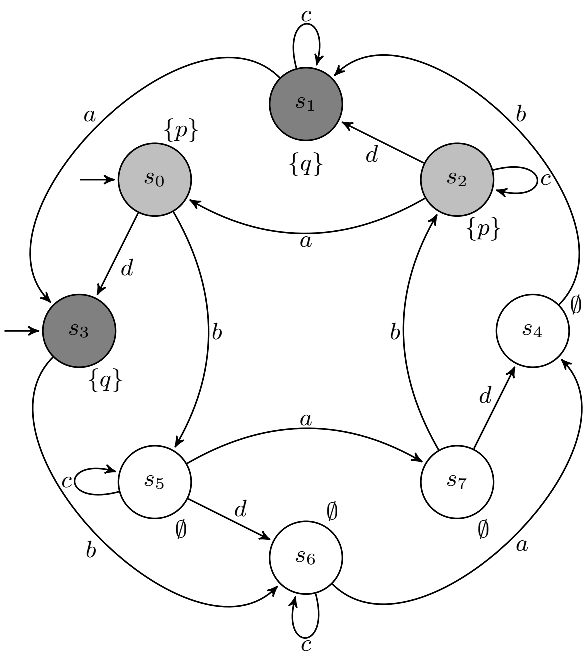
TD 9: Partial Order Reductions

1 Ample Sets

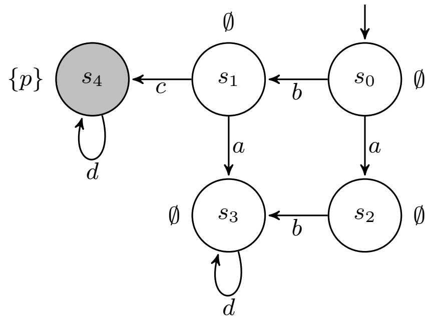
Exercise 1 (Ample Sets)**.** Consider the following transition system with state set $S = \{s_0, \ldots, s_7\}$ and transition alphabet $\Delta = \{a, b, c, d\}$:



1. Compute the independence set $I \subseteq \Delta^2$.
2. What is the set of invisible actions $U \subseteq \Delta$?
3. Propose an assignment $red : S \to 2^{\Delta}$ of ample sets satisfying conditions C_0–C_3 of the lecture notes.
4. Propose a stutter-equivalent system with a reduced set of states.

Exercise 2 (Alternate conditions)**.**

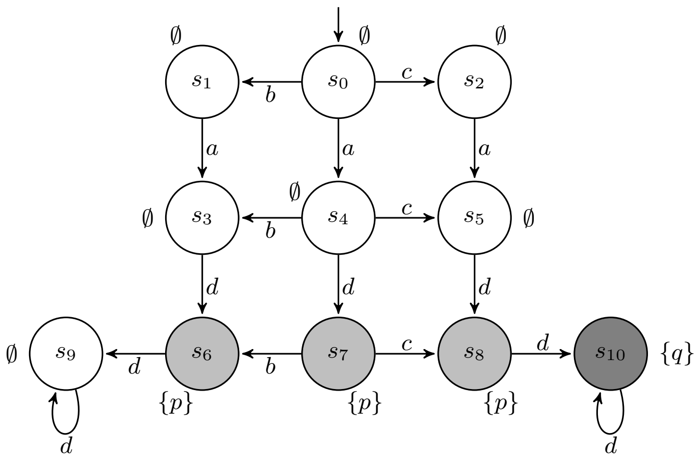
1. Consider the alternate condition C_1': for any s with $red(s) \neq en(s)$, any a in $red(s)$ is independent from every b in $en(s)\backslash red(s)$. Show that C_1 implies C_1'. Does the converse implication hold? *Hint: consider the following system with red:* $s_0 \mapsto \{a\}$, $s_2 \mapsto \{b\}$, *and* $s_3 \mapsto \{d\}$.



2. Consider the alternate condition C_3': any cycle in $\mathcal{K}'$ contains at least one state s with $red(s) = en(s)$. Show that C_0–C_2 and C_3' together imply C_3. Do C_0–C_3 together imply C_3'?

2 CTL(U) Model Checking

Exercise 3 (C_0–C_3 are not Sufficient)**.** Consider the following system with $\Delta = \{a, b, c, d\}$:



1. Let $red(s_0) = \{b, c\}$ and $red(s) = en(s)$ for $s \neq s_0$; show that this ample set assignment is compatible with C_0–C_3.

2. Exhibit a CTL(U) formula that distinguishes between the original system and its reduction.

3. Can you propose an assignment that also complies with C_4: if $red(s) \neq en(s)$, then $|red(s)| = 1$?

3 Nested DFS

Partial order reduction using ample sets is especially suited for on-the-fly algorithms for the emptiness of Büchi automata. The usual, linear-time algorithm for this task uses a *nested depth-first search.*

Recall a DFS-based algorithm for cycle detection from a given state $s \in S$ in a finite directed graph (Q, T), with a global variable $V \subseteq Q$ for the set of already visited vertices:

```
1  found ← false                                  /* no cycle found yet */
2  P ← s                          /* a stack P ∈ Q* of vertices to process */
3  V ← V ∪ {s}                            /* the set of visited vertices */
4  repeat
5      s' ← top(P)
6      if s ∈ T(s') then
7          found ← true
8      else
9          if T(s') \ V ≠ ∅ then
10             s'' ← some(T(s') \ V)      /* some vertice accessible from s' */
11             push(s'', P)
12             V ← V ∪ {s''}
13         else
14             pop(P)
15 until P = ε ∨ found
16 return found
```

Algorithm 1: CYCLE(s)

One way to use this algorithm for Büchi automata emptiness is to first find the accepting states s in F of the automaton $\mathcal{B} = \langle Q, \Sigma, \delta, I, F\rangle$ that are reachable from I (also by an *external* DFS), and then call CYCLE(s) with $V = \emptyset$ for each such state—a quadratic time algorithm. The next exercise refines this approach:

Exercise 4 (Nested DFS)**.** The idea of the nested DFS algorithm is to avoid states from previous cycle searches in later searches—hence the global V in CYCLE. Consider the following external DFS ACYCLE that uses a set of visited states U, and calls CYCLE on reachable accepting states s' of $\mathcal{B}$ *once their reachable states have been processed* (see line 12).

1. Consider a call to ACYCLE(s_0) with empty initial U and V. Assume there exists a call to CYCLE(s) performed by ACYCLE such that, before the call,

$$\text{there is a cycle } q_0 q_1 \cdots q_k,\ q_0 = s = q_k \wedge \exists i,\ q_i \in V\ ; \qquad (\dagger)$$

   without loss of generality assume that s is the first state s.t. (†) occurs. Note that there has to be $s' \in Q$ s.t. CYCLE(s') was invoked before CYCLE(s) and q_i was visited and added to V during this call to CYCLE(s').

   (a) Consider the two cases: s was visited (i.e. pushed on P') before or after s' in the run of ACYCLE, and derive a contradiction in both cases.

```
P′ ← s                              /* a stack P′ ∈ Q* of vertices to process */
U ← U ∪ {s}                                   /* the set of visited vertices */
repeat
    s′ ← top(P′)
    if T(s′) \ U ≠ ∅ then
        s″ ← some(T(s′) \ U)             /* some vertice accessible from s′ */
        push(s″, P′)
        U ← U ∪ {s″}
    else
        pop(P′)          /* all the successors of s′ have been processed */
        if s′ ∈ F then
            found ← Cycle(s′)                            /* call Cycle on s′ */
until P′ = ε ∨ found
```

Algorithm 2: ACycle(s)

(b) Why does ACycle succeeds in finding acceptance cycles from s_0?

2. Provide the missing invocation context for ACycle to solve Büchi automata emptiness.

3. Show that the algorithm works in linear time.

Exercise 5 (Ample Sets in Nested DFS)**.**

1. Assume you are given ample sets for each reachable state (i.e. you can call $red(s)$ for any reachable state s and obtain the ample set for s). Adapt the nested DFS algorithm to only explore the reduced system.

2. Assume now that you are only provided with a $red'(s)$ function that provides ample sets verifying C_0–C_2, but not necessarily C_3. Adapt your algorithm to enforce C'_3 on the fly.