the state explosion problem

Martin Horsch

December 8, 2003

Definition.

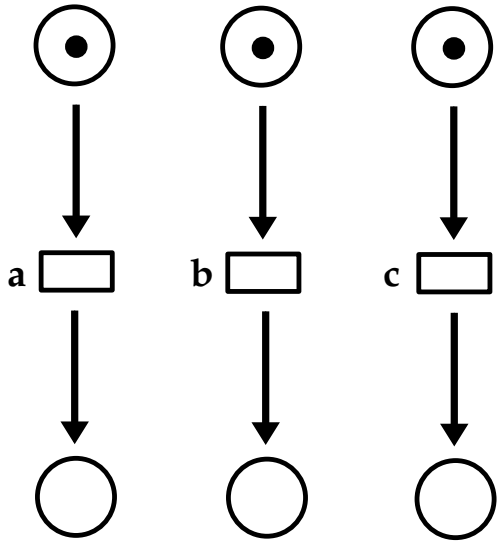
Let $\Pi = (P, T, F, W, m_0)$ be a Petri net such that P is the set of places, T the set of transitions, F the set of arcs and $W : A \to \mathbb{N}$ its weight function. $M : P \to \mathbb{N}$ is the set of markings and $m_0 \in M$ the initial marking on Π. The maximal number of input or output arcs that a place $p \in P$ can have is designated c_F.

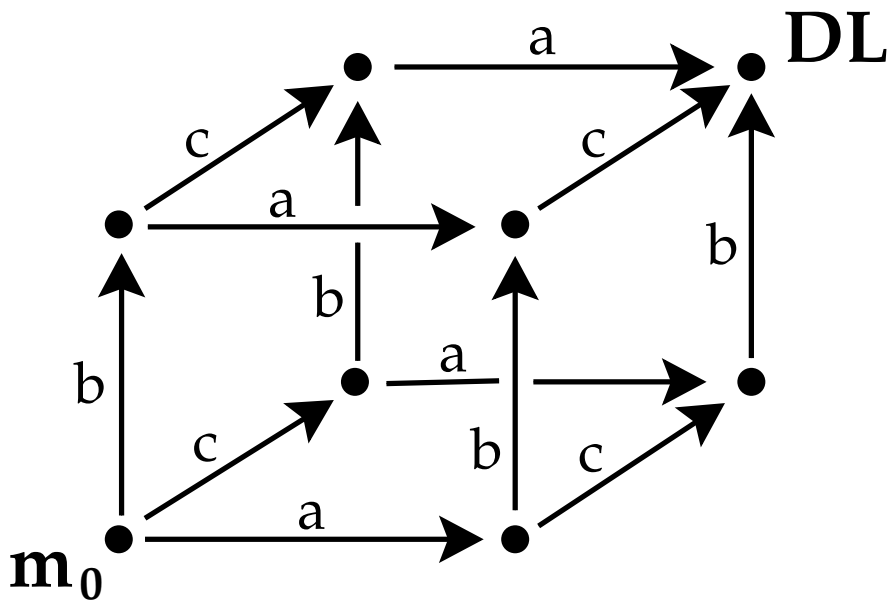
Let $t \in T$. We write $m \to_t$ iff the transition $t \in T$ is enabled at the marking m. Let the marking after the firing of t be m'. For this we write $m \to_t m'$. Such a triple (m, t, m') is called a **semantic transition**.

The **state space** of Π is $\mathcal{R}(\Pi) = (M, T, \Delta, m_0)$ with the set of semantic transitions $\Delta = \{(m, t, m') \in M \times T \times M \mid m \to_t m'\}$. In a state space graph, vertices represent markings and edges represent semantic transitions.

exploding state spaces

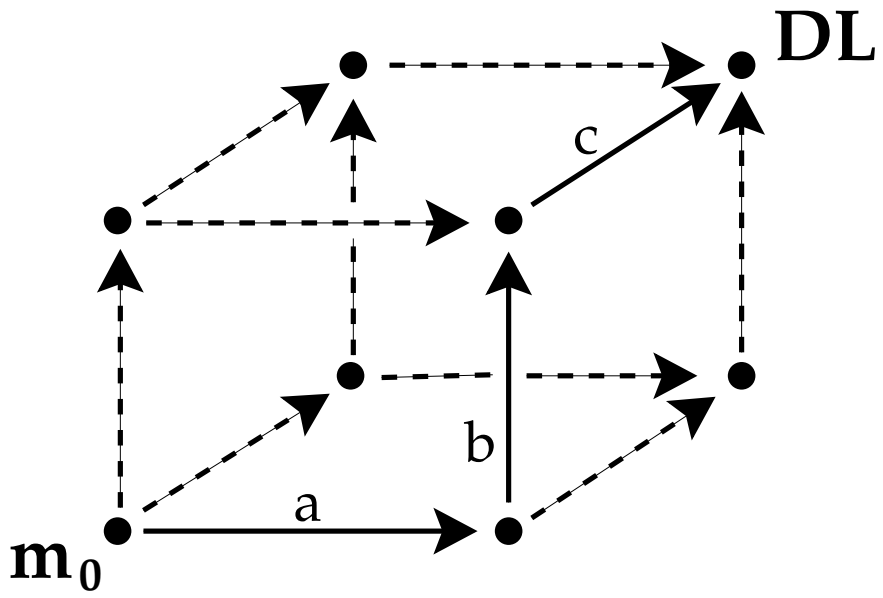
A typical verification problem consists in proving whether a Petri net has a **deadlock**.





A net composed of n parallel nets, each with k reachable components, has a total of k^n states.

In our example net, we can reduce all firing sequences to one without missing the deadlock:



The remaining subset of T is called **stubborn** at m_c. In general, for every marking m_c reached while exploring the state space, we try to eliminate as many transitions as possible.

A simple reachability analysis algorithm would fire all enabled transitions. We hope to achieve a better performance by firing only the stubborn ones.

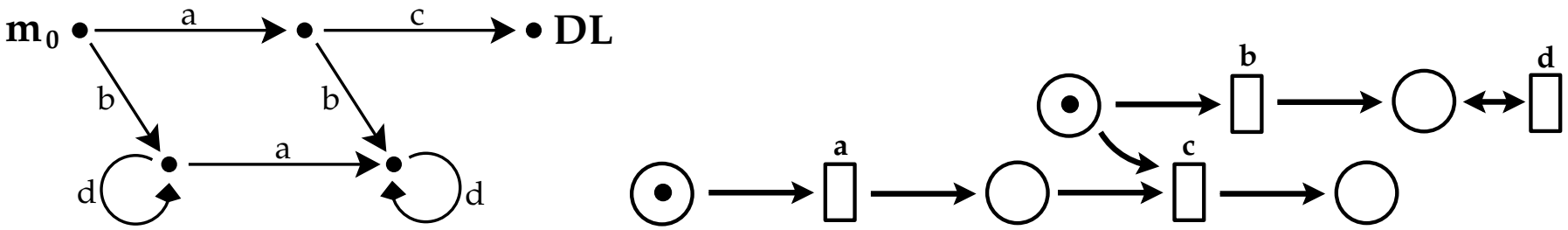
```
reachable ← ∅;  // init

function find_dl(m)
   reachable ← reachable ∪ {m};
   enabled ← the set of enabled transitions at m;
   stubborn ← a stubborn set at m;
   d ← (enabled == ∅);
   foreach transition t in ( stubborn ∩ enabled ) do
      m' ← result of firing t at m;
      if m' ∉ reachable then d ← ( d or find_dl(m') ) fi
return d;
```

The following condition must hold for every stubborn set $T_s \subseteq T$ at a marking m_c.

Condition D1. $\forall \tilde{n} \in M \;\; \forall t \in T_s \;\; \forall \sigma \in (T \setminus T_s)^* :$
$\exists \tilde{m}_c \in M \;\; m_c \rightarrow_\sigma \tilde{m}_c \rightarrow_t \tilde{n} \quad \Longrightarrow \quad \exists n \in M \; m_c \rightarrow_t n \rightarrow_\sigma \tilde{n}$

Is this property sufficient? Consider the following net:



Although a and b are a diamond, only $\{a\}$ is adequate as a stubborn set. By choosing $\{b\}$ a reachability analyzer would miss the deadlock! We must introduce another condition to assure that this problem is resolved.

Definition.

Let $R = (M, T, \Delta, M_0)$ be the state space of a Petri net. Let $m_c \in M$. Then $T_s \subseteq T$ is **dynamically stubborn** at m_c iff it has the following properties:

Condition D1.

$\forall \tilde{n} \in M \;\; \forall t \in T_s \;\; \forall \sigma \in (T \setminus T_s)^* :$

$\exists \tilde{m}_c \in M \;\; m_c \rightarrow_\sigma \tilde{m}_c \rightarrow_t \tilde{n} \quad \Longrightarrow \quad \exists n \in M \; m_c \rightarrow_t n \rightarrow_\sigma \tilde{n}$

Condition D2.

$\exists k \in T_s \;\; \forall \sigma \in (T \setminus T_s)^* : \;\; m_c \rightarrow_\sigma \tilde{m}_c \;\; \Longrightarrow \;\; \tilde{m}_c \rightarrow_k$

A transition is called a **key transition** at m_c iff it qualifies as the transition k in **D2**.

Lemma. Let $R = (M, T, \Delta, m_0)$ be the original state space and $R' = (M, T, \Delta, m_0)'$ the reduced state space. Then there is a deadlock in R' if and only if there is one in R.

Proof.
($\Rightarrow$) The reduction of state space can't create a deadlock, because the only marking where outgoing semantic transitions are removed is m_c. The current marking can't become a deadlock, because at least the key transition remains enabled.

($\Leftarrow$) Let m_d be a deadlock that R can only reach via m_c. Condition $\mathfrak{D}2$ implies that after firing a sequence from $(T \backslash T_s)^*$ the key transition must still be enabled; thus no such sequence can lead from m_c to a deadlock.
Each sequence that leads to a deadlock must therefore be of the type $m_c \rightarrow_{\sigma t \vartheta} m_d$ with $\sigma \in (T \backslash T_s)^*$, $t \in T$, $\vartheta \in T^*$. With $\mathfrak{D}I$ it follows that $m_c \rightarrow_{t \sigma \vartheta} m_d$, a path that remains enabled in the reduced set.

Definition.
Let $R = (M, T, \Delta, M_0)$ be the state space of a Petri net. Let $m_c \in M$. Then $T_s \subseteq T$ is **statically stubborn** at m_c iff the following hold:

S1. $\forall t \in \{t \in T | m_c \rightarrow_t\} : \quad t \in T_s \implies (({\bullet}t){\bullet} \subseteq T_s)$

S2. $\forall t \in \{t \in T | m_c \not\rightarrow_t\} : \quad t \in T_s \Rightarrow \exists p \in {\bullet}t, m_c(p) < W(p,t) : {\bullet}p \subseteq T_s$

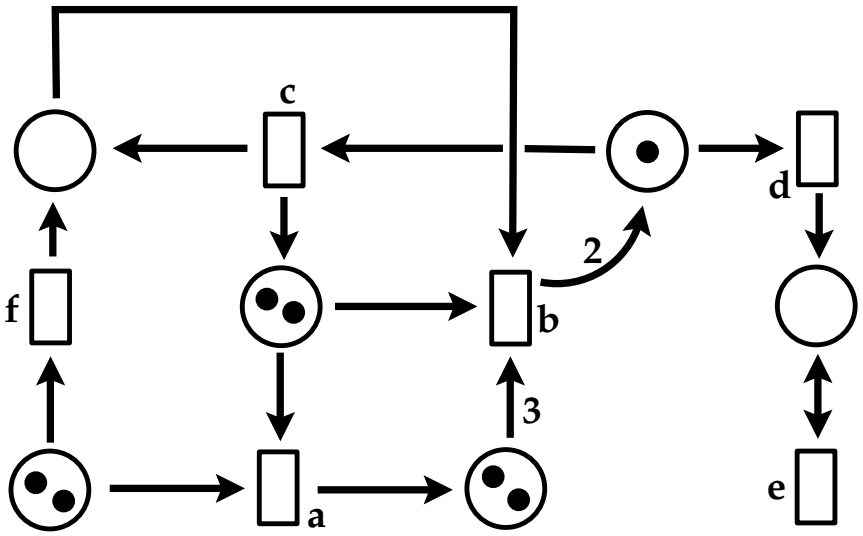
S3. $\exists k \in T_s : \quad m_c \rightarrow_k$

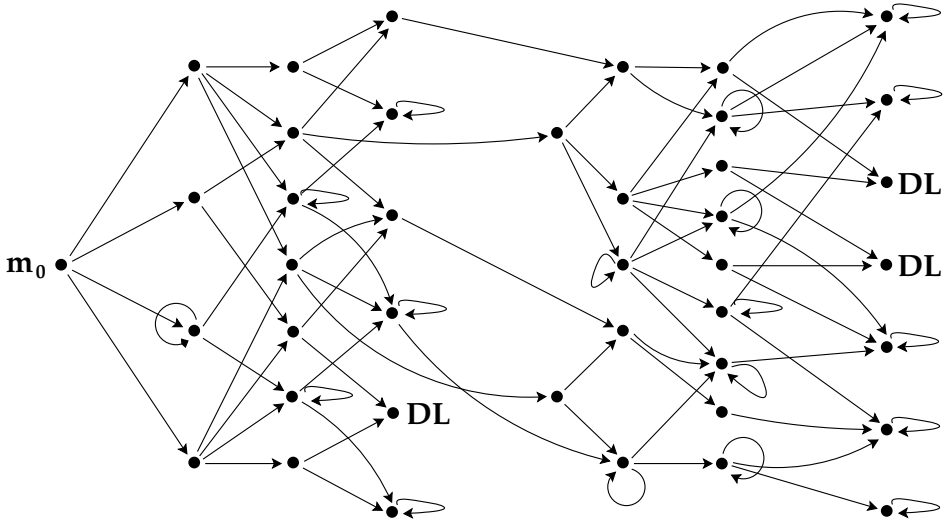
This definition is called static, because it only relies on information available in the reduced state space. **S1** is an implication of the type:

$$\boxed{\textbf{if } t_0 \in T_s \textbf{ then } \{t_1, t_2, \ldots t_n\} \subseteq T_s}$$

Such a structure can be represented by an **implication graph**. Ambiguous implications from **S2** must be resolved at graph construction time.

Find a stubborn set T_s at the initial marking m_0 of this net:





From $\mathfrak{SI}$ the following implications follow:

$a \in T_s \Rightarrow \{b, f\} \subset T_s$, $c \in T_s \Rightarrow d \in T_s$, $d \in T_s \Rightarrow c \in T_s$, $f \in T_s \Rightarrow a \in T_s$,

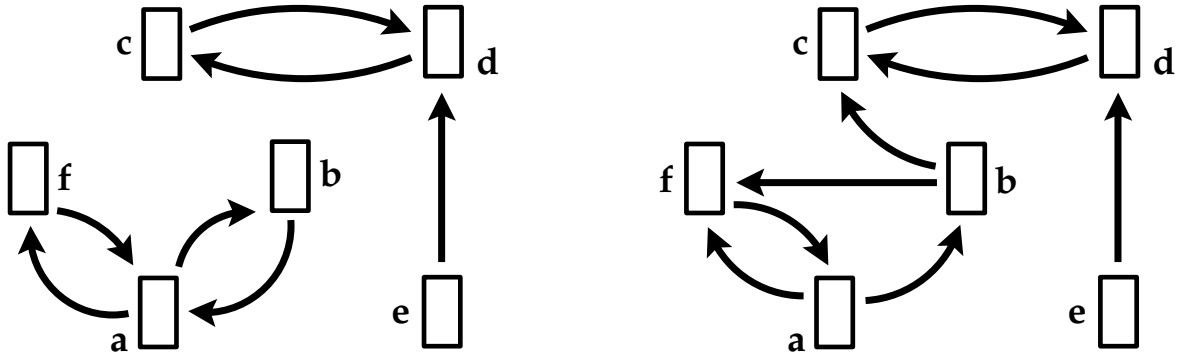
while $\mathfrak{S2}$ allows us to decide at graph construction time whether

$$((b \in T_s \Rightarrow a \in T_s) \ \textbf{or} \ (b \in T_s \Rightarrow \{c, f\} \subset T_s)), \quad e \in T_s \Rightarrow d \in T_s$$

and $\mathfrak{S3}$ requires that at least one $k \in T_s$ be enabled.

algorithms based on implication graphs

Depending on which option an algorithm chooses at $\mathfrak{S}2$, one of the following implication graphs is constructed:



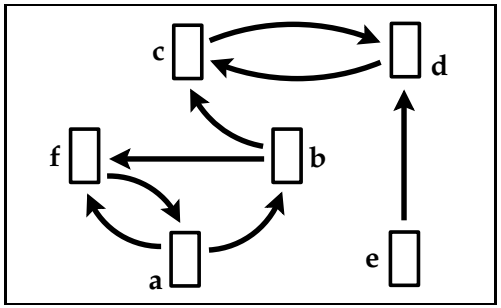
The **closure algorithm** starts with the first enabled transition and then extends the stubborn set by induction in $\mathcal{O}((c_F)^2\,|T|) \subset \mathcal{O}(|T|^2)$. In our example, this leads to the possible results:

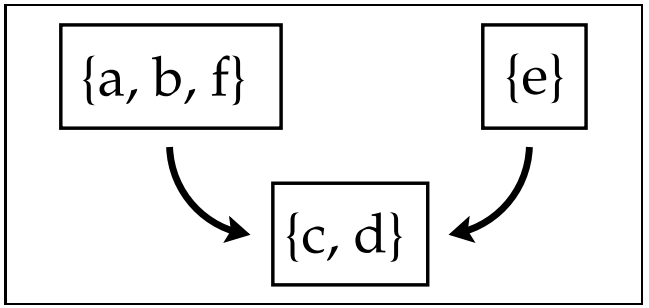
graph 1: $\quad a \in T_S \Rightarrow \{a, \underline{b}, \underline{f}\} = T_S$

graph 2: $\quad a \in T_S \Rightarrow \{a, \underline{b}, \underline{f}\} \subseteq T_S \Rightarrow \{a, b, \underline{c}, f\} \subseteq T_S \Rightarrow \{a, b, c, \underline{d}, f\} = T_S$

Strongly connected components of a directed graph (V, E) are subsets $V' \subseteq V$ such that for all $u, v \in V'$ there is a path from u to v as well as vice versa.

The **maximal strongly connected component algorithm** partitions the implication graph into sets of this type. Then it returns the union of one of the lowest components that contain an enabled transition and all components below, in this case $T_s = \{c, d\}$:





With a modified depth first search for finding the strong components, this algorithm terminates in $\mathcal{O}(|T|^2)$ just like the closure algorithm.

Tarjan's algorithm, a modified depth first search, determines the maximal strongly connected components of a graph in $O(|V| + |E|)$:

```
dfs ← 0;  visited ← ∅;  // init

function visit(k)
   visited ← visited ∪ {k};
   min ← dfs;  dfs ← dfs+1;
   push k on the stack;
   foreach node c of in (children(k)\visited) do
      m ← visit(c);
      if m < min then min ← m fi od;
   if min = k then
      strcon ← ∅;
      loop do
         pop t from the stack;  strcon ← strcon ∪ {t};
         if t == k then break fi od
      output strcon fi;
return min
```

partial deletion of $\wedge$-$\vee$-graphs

In many cases we can get a better performance if we relax condition $\mathfrak{SI}$ and change $\mathfrak{S3}$ to guarantee the key transition property.

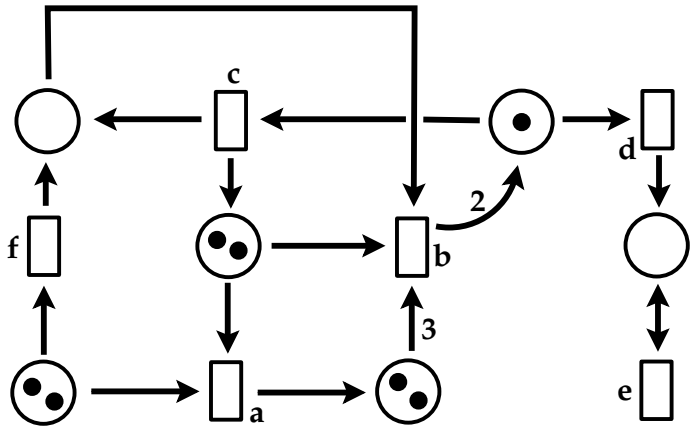
$\mathfrak{SI}_{lax}$. $\forall t \in \{t \in T | m_c \rightarrow_t\} : \quad t \in T_s \implies ((\bullet t)\bullet \subseteq T_s) \vee (\bullet(\bullet t) \subseteq T_s)$

$\mathfrak{S2}$. $\forall t \in \{t \in T | m_c \not\rightarrow_t\} : \; t \in T_s \Rightarrow \exists p \in \bullet t, m_c(p) < W(p,t) : \bullet p \subseteq T_s$

$\mathfrak{S3}_{ext}$. $\exists k \in T_s : \; m_c \rightarrow_k \wedge (\bullet k)\bullet \subseteq T_s$

These requirements for a stubborn set can be represented best by an $\wedge$-$\vee$-graph with different types of nodes for conjunctions and disjunctions. This method avoids the problem of deciding at graph generation time which side of a disjunction should be active.

We apply the changed set of conditions to our example net:



$\mathfrak{SI}_{lax} \implies$ if $a \in T_s$ then $c \in T_s \vee b \in T_s$

$\mathfrak{S2} \implies$ if $b \in T_s$ then $a \in T_s \vee \{c, f\} \subseteq T_s$

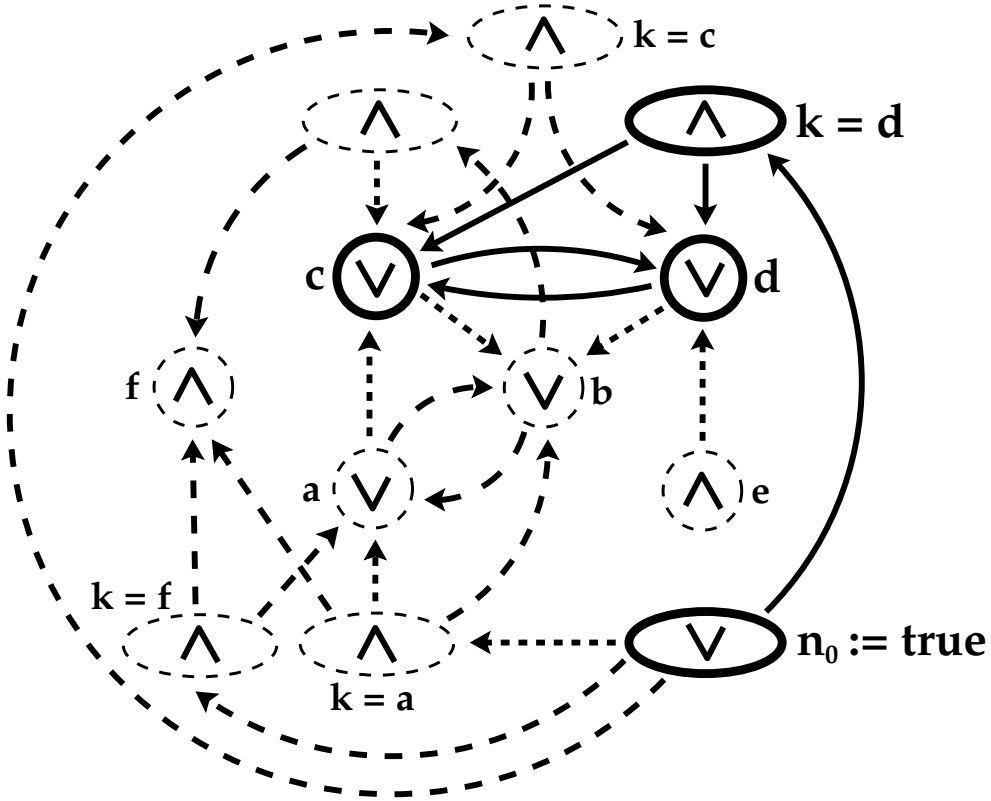
$\mathfrak{SI}_{lax} \implies$ if $c \in T_s$ then $b \in T_s \vee d \in T_s$

$\mathfrak{SI}_{lax} \implies$ if $d \in T_s$ then $b \in T_s \vee c \in T_s$

$\mathfrak{S2} \implies$ if $e \in T_s$ then $d \in T_s$

$\mathfrak{S3}_{ext} \implies$ T_s must contain an enabled key transition k while
$(k = a) \Rightarrow \{b, f\} \subset T_s, \quad (k = c) \Rightarrow d \in T_s$
$(k = d) \Rightarrow c \in T_s, \quad$ and $\quad (k = f) \Rightarrow a \in T_s.$

This corresponds to the following $\wedge$-$\vee$-graph:



By disabling as many nodes as possible, the **deletion algorithm** finds $T_s = \{c, d\}$, the minimal statically stubborn set at m_0.