Solution

Petri nets – Homework 8

Discussed on Thursday 2nd July, 2015.

For questions regarding the exercises, please send an email to meyerphi@in.tum.de or just drop by at room 03.11.042.

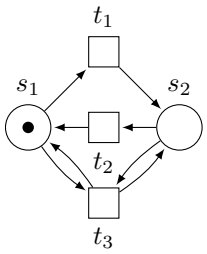
Exercise 8.1 Commoner's Liveness Theorem for general Petri nets

(a) Exhibit a non-live Petri net (N, M_0) where every proper siphon contains a trap marked at M_0.

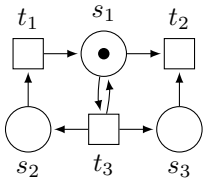
(b) Exhibit a live Petri net (N, M_0) with a proper siphon R of N that does not contain a trap marked at M_0.

Solution:

(a) The following Petri net is non-live, as t_3 is never enabled, and has $\{s_1, s_2\}$ as its only proper siphon, which contains the trap $\{s_1, s_2\}$ marked at M_0.



(b) The following Petri net is live, and the siphon $R = \{s_1, s_2, s_3\}$ contains no trap initially marked, as the net has no proper traps.



Exercise 8.2 Hack's Boundedness Theorem for general Petri nets

A structural component of Petri nets are S-components (Definition 5.3.5 in the script):

Definition 8.2.1. [S-component] Let $N = (S, T, F)$ be a net. A subnet $N' = (S', T', F')$ of N is an *S-component* of N if

1. $T' = {}^\bullet S' \cup S'^\bullet$ (where ${}^\bullet s = \{t \in T \mid (t, s) \in F\}$, and analogously for $s^\bullet$).
2. N' is a strongly connected S-net.

The fundamental propery of S-components is (Proposition 5.3.6 in the script):

Proposition 8.2.1. Let (N, M_0) be a Petri net and let $N' = (S', T', F')$ be an S-component of N. Then $M_0(S') = M(S')$ for every marking M reachable from M_0.

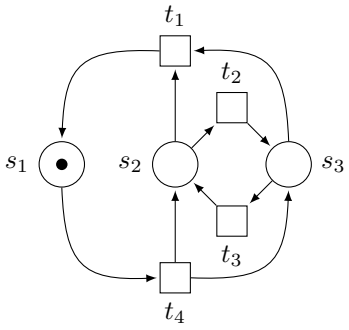
(a) Prove: Let (N, M_0) be a Petri net. If every place of N belongs to an S-component, then (N, M_0) is bounded.

(b) Exhibit a live and bounded Petri net (N, M_0) with a place s of N that does not belong to any S-component.

Solution:

(a) Let s be a place of N and $N' = (S', T', F')$ an S-component with $s \in S'$. For all reachable markings M, we have $M(s) \leq M(S') = M_0(S)$, so s is bounded. As all places are bounded, (N, M_0) is also bounded.

(b) The following Petri net is live and bounded, but s_2 does not belong to any S-component $N' = (S', T', F')$, as that would imply $t_1, t_2, t_3, t_4 \in T'$, which further implies $s_3 \in S'$, but then N' is not an S-net.



Exercise 8.3 Minimal path length in 1-bounded Petri nets

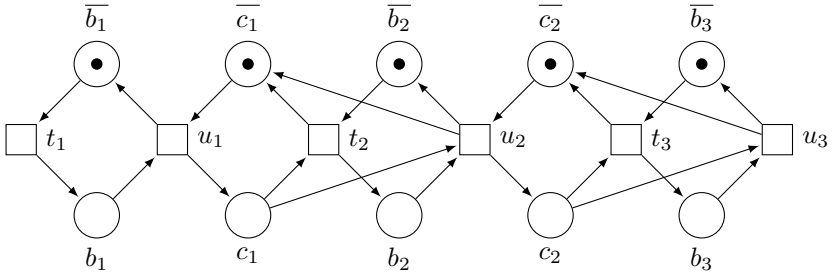
For each $n \in \mathbb{N}$, give a live and 1-bounded Petri net (N, M_0) and a marking M of N such that the size of the net grows linearly, but the length of the minimal occurrence sequence σ with $M_0 \xrightarrow{\sigma} M$ grows exponentially, i.e., $|\sigma| \in \Omega(2^n)$.

Hint: Try to model a counter with a binary encoding that counts incrementally from 0 to 2^n by firing one or more transitions for each step.

Solution:

We encode a binary number $b = b_1 b_2 \dots b_n$ with the places b_i and $\overline{b_i}$ for $1 \leq i \leq n$ to denote if b_i is 0 or 1. We also add places c_i and $\overline{c_i}$ for $1 \leq i \leq n-1$ to denote if there is a carry bit from b_i to $b_i + 1$. We add transitions to increase each b_i if $i = 1$ or there is a carry at $i - 1$. The transition either moves the token from b_i to $\overline{b_i}$ or resets it and adds a carry to c_i. The initial marking M_0 puts a token on each $\overline{b_i}$ and $\overline{c_i}$ and the target marking M puts a token on each b_i and $\overline{c_i}$. The initial marking represents the number $b = 0$ and the final marking $b = 2^n - 1$, and each transition increases b by at most 1, resulting in a minimal path length of at least $2^n - 1$.

For example, for $n = 3$, we have the following Petri net:



Exercise 8.4 Reducing SAT to reachability in free-choice systems

Reduce the satisfiability problem for boolean formulas in conjunctive normal form to the reachability problem in free-choice systems.

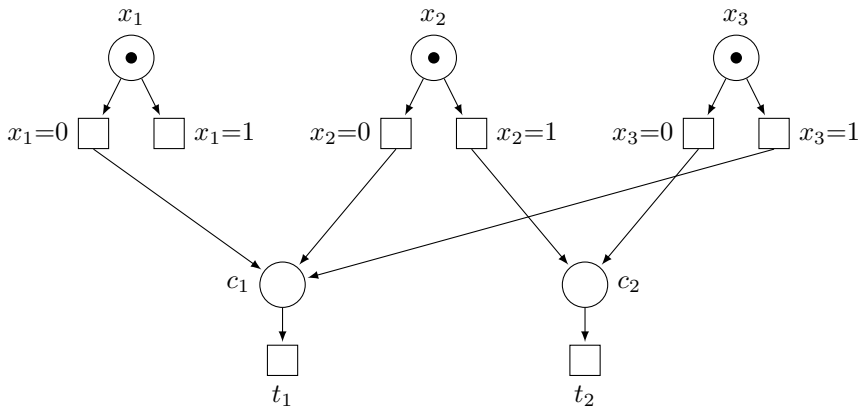
For that, give a polynomial time translation that, for a given formula φ, produces a free-choice system (N, M_0) and a marking M such that φ is satisfiable iff M is reachable in (N, M_0). Describe your reduction informally and give the resulting Petri net when applying it to the formula below.

$$\varphi = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$

Solution:

We use places x_i for each variable and transitions to choose their assignment. Places c_j for each clause get marked if the assignment makes that clause true. A transition for each clause can remove additional tokens from the clause places. The target marking M is given by $M(c_j) = 1$ for each clause place c_j and $M(s) = 0$ for all other places s. M is reachable if and only if all clauses can be made true by an assignment of the variables, i.e., φ is satisfiable.

For the given formula, we get the following free-choice system and the target marking with one token in c_1 and c_2 each and no tokens elsewhere.



Exercise 8.5 **Simulating a bounded stack**

A *bounded stack* is a tuple $K = (\Gamma, k)$, where Γ is the stack alphabet and k is the stack size. A *configuration* of the stack is a sequence $\gamma \in \Gamma^*$. On a stack, we can perform the following actions:

- Push an element $c \in \Gamma$ on the top of the stack if the stack has less than k elements.
- Pop an element $c \in \Gamma$ from the stack if the top element is c.
- Assert that the stack is empty if the stack has no elements.

More formally, for a bounded stack (Γ, k), we have the *actions* $A_\Gamma = \{push_c, pop_c \mid c \in \Gamma\} \cup \{empty\}$ which induce the following transitions rules on the configurations:

$$\begin{aligned} \epsilon &\xrightarrow{empty} \epsilon \\ \gamma &\xrightarrow{push_c} \gamma c \quad \text{for } c \in \Gamma,\ \gamma \in \Gamma^*,\ |\gamma| < k \\ \gamma c &\xrightarrow{pop_c} \gamma \quad \text{for } c \in \Gamma,\ \gamma \in \Gamma^* \end{aligned}$$

A sequence of actions $w = a_1 a_2 \dots a_n \in A_\Gamma^*$ is a *computation* of a stack K if there exist configurations $\gamma_1 \gamma_2 \dots \gamma_n$ such that $\epsilon \xrightarrow{a_1} \gamma_1 \xrightarrow{a_2} \gamma_2 \dots \gamma_{n-1} \xrightarrow{a_n} \gamma_n$ is a transition sequence according to the above transition rules. For example, with $K = (\{a, b\}, 2)$, the sequence *empty* $push_a$ $push_b$ pop_b pop_a *empty* $push_b$ is a computation with the intermediate configurations

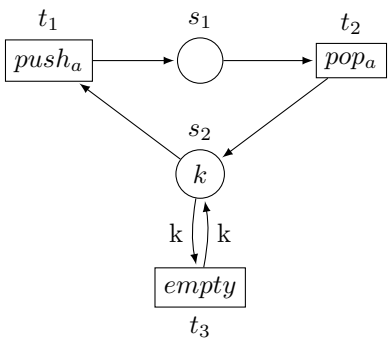
$$\epsilon \xrightarrow{empty} \epsilon \xrightarrow{push_a} a \xrightarrow{push_b} ab \xrightarrow{pop_b} a \xrightarrow{pop_a} \epsilon \xrightarrow{empty} \epsilon \xrightarrow{push_b} b.$$

A Petri net (N, M_0) together with a labeling function $h_l : T \to A_\Gamma \cup \{\tau\}$ *simulates* the computations of a bounded stack $K = (\Gamma, k)$ if, with the homomorphism $h : T^* \to A_\Gamma^*$ defined by

$$\begin{aligned} h(\epsilon) &= \epsilon \\ h(t) &= \epsilon \qquad \text{if } h_l(t) = \tau \\ h(t) &= h_l(t) \qquad \text{if } h_l(t) \in A_\Gamma \\ h(\sigma t) &= h(\sigma) h(t), \end{aligned}$$

we have: $\{h(\sigma) \mid \exists M : M_0 \xrightarrow{\sigma} M\} = \{w \in A_\Gamma^* \mid w \text{ is a computation of } K\}$. Basically, some transitions of the Petri net correspond to actions of the stack, and the occurrence sequences of the net projected onto the actions of these transitions correspond to the computations of the stack.

As an instance, if $\Gamma = \{a\}$, for each k, we can give the following Petri net (with weights) which simulates a stack $K = (\{a\}, k)$. The label of each transition is given inside the box for the transition.



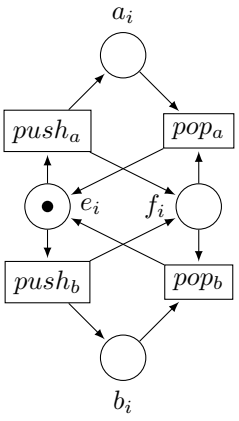
For $\Gamma = \{a, b\}$ and a given k, give a construction to produce a Petri net that simulates the bounded stack $K = (\{a, b\}, k)$. Give the resulting Petri net when applying the construction with $k = 3$. Describe informally how the Petri net changes as k increases. Ensure that the size of the Petri net only grows linearly with k.

Solution:

For each position $1 \leq i \leq k$ of the stack, we represent the state with four places: a_i and b_i, which are marked if there is an a or b in this position, and e_i and f_i, which are marked if the position is empty or full, respectively.

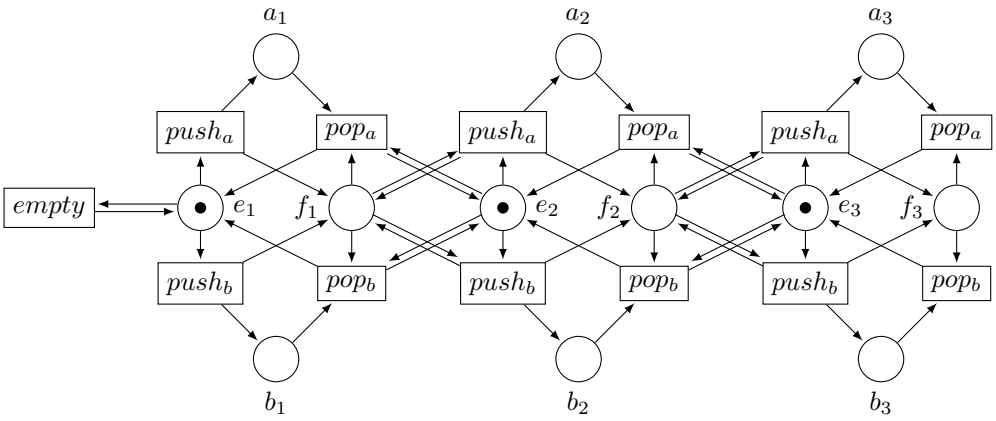
Then we add transitions with the labels $push_a$, pop_a, $push_b$ and pop_b, which either push the corresponding element if the position is empty, or pop it if the element is at that position and the position is full.

With that, for one position, we get the following subnet. The actual names of the transitions are irrelevant and not shown.



When combining several positions, we need to ensure that elements are pushed and pulled on and from the top of the stack. A transition labeled $push_c$ at position i should only be enabled if i is empty and either $i = 1$ or position $i - 1$ is full. A transition labeled pop_c at position i should only be enabled if i is full, the element c is at that position and either $i = k$ or position $i + 1$ is empty. That way, if position i is empty, then for all $j > i$, j is also empty, and if position i is full, then for all $j < i$, j is also full.

Finally, we add a transition labeled *empty*, which checks if position 1 is empty. This gives us the following net for $k = 3$:



An alternative solution is to encode the index of the top of the stack with places s_i for $0 \leq i \leq k$. We can only push or pop at a certain position if it is the top of the stack. This gives us the following solution for $k = 3$: