TD 12: Petri Net Unfoldings, Pushdown Systems

Exercise 1 (Comparison)**.** Let us compare the reduction technique based on ample sets with Petri net unfoldings. We shall see that both have advantages and disadvantages over each other.

For a Petri net N, let $\mathcal{U}(N)$ be its unfolding and $\mathcal{M}(N)$ be the associated transition system in which, for simplicity, we assume all actions to be invisible, and that the independence relation used for reduction is maximal.

1. First, construct a Petri net N with two transitions a, b such that: (i) the input places of a and b overlap; (ii) a and b are independent in $\mathcal{M}(N)$.

In the following, let $(N_k)_{k\geq 1}$ be a family of 1-safe Petri nets such that for all k, the size of N_k is $\mathcal{O}(k)$.

2. Construct a family of nets such that for all k, any complete prefix of $\mathcal{U}(N_k)$ is at least of size 2^k, but $\mathit{red}(\mathcal{M}(N_k))$ is of size $\mathcal{O}(k)$.
3. Construct a family of nets such that for all k, $\mathit{red}(\mathcal{M}(N_k))$ is at least of size 2^k, but there is a complete prefix of $\mathcal{U}(N_k)$ of size $\mathcal{O}(k)$.

Hint: It suffices to regard nets whose rechability graph is acyclic. For (3), try to construct N_k from k separate components such that $\mathcal{U}(N_k)$ is simply the juxtaposition of the unfoldings of the components.

Exercise 2 (Adequate Partial Orders)**.** A partial order $\prec$ between events is *adequate* if the three following conditions are verified:

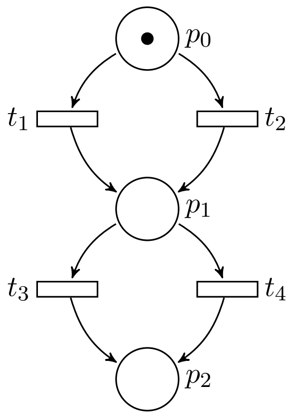
(a) $\prec$ is well-founded,

(b) $\lfloor t \rfloor \subsetneq \lfloor t' \rfloor$ implies $t \prec t'$, and

(c) $\prec$ is preserved by finite extensions: as in the lecture notes, if $t \prec t'$ and $B(t) = B(t')$, and E and E' are two isomorphic extensions of $\lfloor t \rfloor$ and $\lfloor t' \rfloor$ with $\lfloor u \rfloor = \lfloor t \rfloor \oplus E$ and $\lfloor u' \rfloor = \lfloor t' \rfloor \oplus E'$, then $u \prec u'$.

As you can guess, adequate partial orders result in complete unfoldings. (An event e is a cutoff if there exists $f \prec e$ such that the markings associated with e and f are the same.)

1. Show that $\prec_s$ defined by $t \prec_s t'$ iff $|\lfloor t \rfloor| < |\lfloor t' \rfloor|$ is adequate.
2. Construct the finite unfolding of the following Petri net using $\prec_s$; how does the size of this unfolding relate to the number of reachable markings?

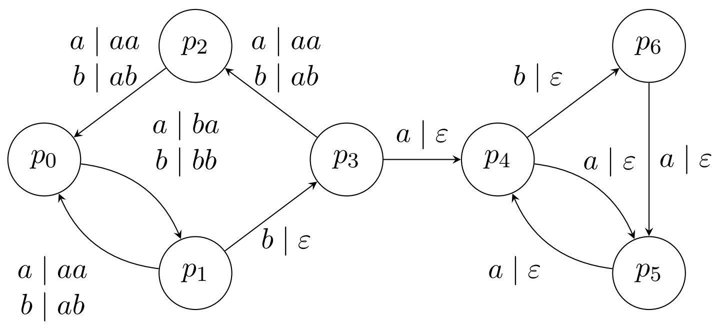


3. Suppose we define an arbitrary total order $\ll$ on the transitions T of the Petri net, i.e. they are $t_1 \ll \cdots \ll t_n$. Given a set S of events and conditions of $\mathcal{Q}$, $\varphi(S)$ is the sequence $t_1^{i_1} \cdots t_n^{i_n}$ in T^* where i_j is the number of events labeled by t_j in S. We also note $\ll$ for the lexicographic order on T^*.

   Show that $\prec_e$ defined by $t \prec_e t'$ iff $|\lfloor t \rfloor| < |\lfloor t' \rfloor|$ or $|\lfloor t \rfloor| = |\lfloor t' \rfloor|$ and $\varphi(\lfloor t \rfloor) \ll \varphi(\lfloor t' \rfloor)$ is adequate. Construct the finite unfolding for the previous Petri net using $\prec_e$.

4. There might still be examples where $\prec_e$ performs poorly. One solution would be to use a *total* adequate order; why? Give a 1-safe Petri net that shows that $\prec_e$ is not total.

Exercise 3 (Computing $pre^*(C)$)**.** Consider the pushdown system represented below, with stack alphabet $\Gamma = \{a, b\}$.



Apply the algorithm described in the lecture notes to compute a $\mathcal{P}$-automaton accepting $pre^*(p_6 b^*)$.

Exercise 4 (Labelled Pushdown Systems)**.** Let $\mathcal{P} = (P, \Gamma, \Delta, \Sigma)$ be a labelled pushdown system, i.e. the rules in Δ are of the form $pA \overset{a}{\hookrightarrow} qw$, where $p, q \in P$ are control locations,

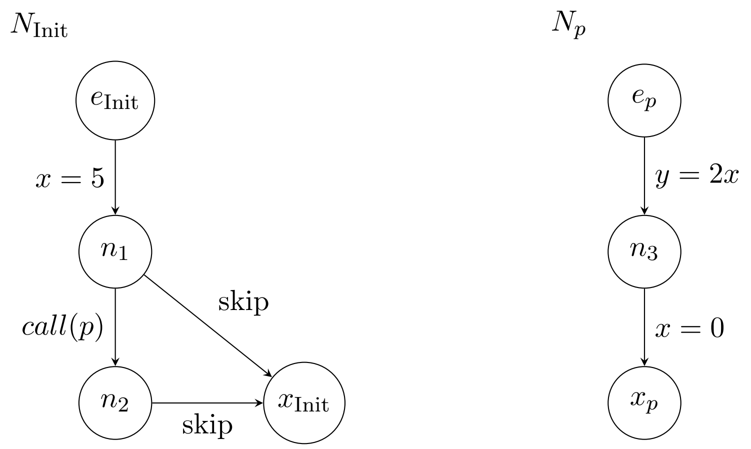
$A \in \Gamma$ and $w \in \Gamma^*$ are stack symbols, and additionally $a \in \Sigma$ is an *action*. The set of configurations $Con(\mathcal{P})$ consists of the tuples qw with $q \in P$ and $w \in \Gamma^*$. For two configurations c, c' we write $c \overset{w}{\Rightarrow} c'$, where $w \in \Sigma^*$, if c can be transformed into c' by a sequence of rules whose labels yield w.

Given a regular set of configurations C, it is known how to compute $pre^*(C) = \{ c \in Con(\mathcal{P}) \mid \exists c' \in C, w \in \Sigma^* : c \overset{w}{\Rightarrow} c' \}$. If C is accepted by an automaton with n states, this takes $\mathcal{O}(n^2 \cdot |\Delta|)$ time.

1. Let $L \subseteq \Sigma^*$ be a regular language and C be a regular set of configurations. We define
$$pre^*[L](C) := \{ c \in Con(\mathcal{P}) \mid \exists c' \in C, w \in L : c \overset{w}{\Rightarrow} c' \}.$$
One can prove that $pre^*[L](C)$ is regular. Describe how to compute a finite automaton accepting $pre^*[L](C)$.

2. Give a bound on the amount of time it takes to compute $pre^*[L](C)$.

Exercise 5 (Data-flow Analysis)**.** We consider a problem from interprocedural data-flow analysis. A program consists of a set *Proc* of procedures that can execute and recursively call one another. The behaviour of each procedure p is described by a flow graph, an example with two procedures is shown below.



Formally, a flow graph for procedure $p \in Proc$ is a tuple $G_p = (N_p, A, E_p, e_p, x_p)$, where

- N_p are the nodes, corresponding to program locations; we denote $N := \bigcup_{p \in Proc} N_p$.
- $A = A_I \cup \{ call(p) \mid p \in Proc \}$ are the actions, where A_I are *internal actions* (such as assignments etc); additionally an action can call some procedure. A is identical for all procedures.

- $E_p \subseteq N_p \times A \times N_p$ are the edges, labelled with actions from A. We denote $E := \bigcup_{p \in Proc} E_p$.
- e_p is the *entry point* of procedure p, i.e. when p is called, execution will start at e_p.
- x_p is the *exit point* of p (without any outgoing edges); when x_p is reached, p terminates and execution resumes at last call site of p.

1. Construct a labelled pushdown system with one single control location that expresses the behaviour of the procedures in *Proc*.

Suppose that the internal actions in A_I describe assignments to global variables, i.e. they are of the form $v := expr$, where v is a variable and *expr* the right-hand-side expression. If v is a variable, then $D_v \subseteq A_I$ is the set of actions that assign a value to v and $R_v \subseteq A_I$ the set of actions where v occurs on the right-hand side.

Let $Init \in Proc$ be an initial procedure and $n \in N$ a node in the flow graph. We say that variable v is *live* at n if there exists a node n' and an execution that (i) starts at e_{Init}, (ii) passes n, (iii) finally reaches n' with an action from R_v, and (iv) there is no assignment to v between n and n' in this execution. (Intuitively, this means that the value that v has at n matters for some execution; this is used in compiler construction to determine whether an optimizing compiler may "forget" the value of v at n.) For instance, in the shown example, the variable x is live at n_1 and e_p, but not in the other nodes.

2. Describe a regular language $L \subseteq A^*$ that describes the sequences of actions that can happen along such executions between n and n'.
3. Describe how, given a variable v, one can compute the set of nodes n such that v is live at n.