TD 9: Pushdown Systems

Reminder:

A *pushdown system (PDS)* is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where P is a finite set of *control states*, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of *rules*. We write $pA \hookrightarrow qw$ when $((p, A), (q, w)) \in \Delta$. We associate with a PDS $\mathcal{P}$ and an initial configuration $c_0 \in P \times \Gamma^*$ the transition system $\mathcal{T}_\mathcal{P} = (Con(\mathcal{P}), \rightarrow, c_0)$, where $Con(\mathcal{P}) = P \times \Gamma^*$ is the set of *configurations*, and $pAw' \rightarrow qww'$ for all $w' \in \Gamma^*$ iff $pA \hookrightarrow qw \in \Delta$. We write $pw \Rightarrow p'w'$ if there is a path from pw to $p'w'$ in $\mathcal{T}_\mathcal{P}$.

Let $\mathcal{P}$ be a PDS. A $\mathcal{P}$*-automaton* is a finite automaton $\mathcal{A} = (Q, \Gamma, P, T, F)$, where the alphabet of $\mathcal{A}$ is the stack alphabet Γ, and the initial states of $\mathcal{A}$ are the control states P. It is *normalized* if there are no transitions leading into initial states. We say that $\mathcal{A}$ *accepts* the configuration pw if $\mathcal{A}$ has a path labelled by input w starting at p and ending at some final state. We denote by $\mathcal{L}(\mathcal{A})$ be the set of configurations accepted by $\mathcal{A}$. A set C of configurations is called *regular* if there is some $\mathcal{P}$-automaton $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = C$.

Given a set C of configurations of $\mathcal{P}$, we let

$$pre^*(C) = \{c' \mid \exists c \in C : c' \Rightarrow c\}$$
$$post^*(C) = \{c' \mid \exists c \in C : c \Rightarrow c'\}$$

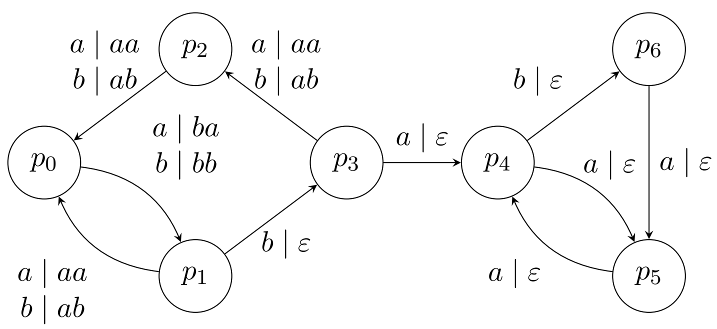
If C is regular, then so are $pre^*(C)$ and $post^*(C)$.

If $\mathcal{A}$ is a normalized $\mathcal{P}$-automaton accepting C, $\mathcal{A}$ can be transformed into an automaton accepting $pre^*(C)$ by applying the following saturation rule until no transition can be added:

> If $q \xrightarrow{w} r$ currently holds in $\mathcal{A}$ and $pA \hookrightarrow qw$ is a rule in $\mathcal{P}$, then add the transition (p, A, r) to $\mathcal{A}$.

The procedure for $post^*(C)$ is similar.

Exercise 1 (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 2 (Dickson's Lemma)**.** A *quasi-order* $(A, \leq)$ is a set A endowed with a reflexive and transitive ordering relation $\leq$. A *well quasi order* (wqo) is a quasi order $(A, \leq)$ s.t., for any infinite sequence $a_0 a_1 \cdots$ in A^ω, there exist indices $i < j$ with $a_i \leq a_j$.

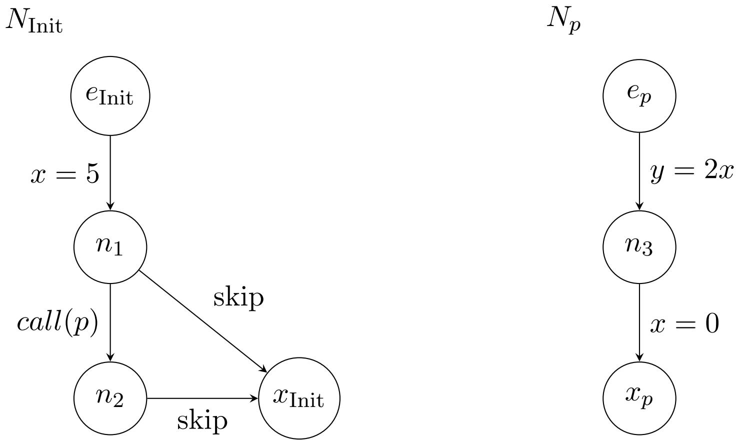
1. Let $(A, \leq)$ be a wqo and $B \subseteq A$. Show that $(B, \leq)$ is a wqo.
2. Show that $(\mathbb{N} \uplus \{\omega\}, \leq)$ is a wqo.
3. Let $(A, \leq)$ be a wqo. Show that any infinite sequence $a_0 a_1 \cdots$ in A^ω embeds an infinite increasing subsequence $a_{i_0} \leq a_{i_1} \leq a_{i_2} \leq \cdots$ with $i_0 < i_1 < i_2 < \cdots$.
4. Let $(A, \leq_A)$ and $(B, \leq_B)$ be two wqo's. Show that the cartesian product $(A \times B, \leq_\times)$, where the product ordering is defined by $(a, b) \leq_\times (a', b')$ iff $a \leq_A a'$ and $b \leq_B b'$, is a wqo.

Exercise 3 (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 4 (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.