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

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

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

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

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

If $A$ is a normalized $\mathcal{P}$-automaton accepting $C$, $A$ 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 $A$ and $pA \rightarrow qw$ is a rule in $\mathcal{P}$, then add the transition $(p, A, r)$ to $A$.

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

Exercise 1 (Computing $pre^*(C)$). Consider the pushdown system represented below, with stack alphabet $\Gamma = \{a, b\}$.

![Diagram of pushdown system](image)
Apply the algorithm described above to compute a $\mathcal{P}$-automaton accepting $\text{pre}^*(p_0b^*)$.

**Exercise 2** (Labelled Pushdown Systems). Let $\mathcal{P} = (P, \Gamma, \Delta, \Sigma)$ be a labelled pushdown system, i.e. the rules in $\Delta$ are of the form $pA \xrightarrow{w} qw$, 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. For two configurations $c, c'$ we write $c \xrightarrow{w} c'$, 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 $\text{pre}^*(C) = \{ c \in \text{Con}(\mathcal{P}) \mid \exists c' \in C, w \in \Sigma^* : c \xrightarrow{w} c' \}$. If $C$ is accepted by an automaton with $n$ states, this takes $O(n^2 \cdot |\Delta|)$ time.

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

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

**Exercise 3** (Data-flow Analysis). We consider a problem from interprocedural data-flow analysis. A program consists of a set $\text{Proc}$ 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 $p \in \text{Proc}$ 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 $N := \bigcup_{p \in \text{Proc}} N_p$. 

\[
\begin{align*}
N_{\text{init}} & \quad N_p \\
&
\begin{array}{c}
\quad e_{\text{init}} \\
\quad \text{xinit} \\
n_1 \quad \text{call}(p) \quad \text{skip} \\
n_2 \quad \text{skip} \\
n_3 \quad x = 0 \\
e_p \quad y = 2x \\
x_p
\end{array}
\end{align*}
\]
• \( A = A_I \cup \{ \text{call}(p) \mid p \in \text{Proc} \} \) 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 \( E := \bigcup_{p \in \text{Proc}} E_p \).

• \( e_p \) 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 \( \text{Proc} \).

Suppose that the internal actions in \( A_I \) describe assignments to global variables, i.e. they are of the form \( v := \text{expr} \), where \( v \) is a variable and \( \text{expr} \) 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 \( \text{Init} \in \text{Proc} \) 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 \( e_{\text{Init}} \), (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.

1. Describe a regular language \( L \subseteq A^* \) that describes the sequences of actions that can happen along such executions between \( n \) and \( n' \).

2. Describe how, given a variable \( v \), one can compute the set of nodes \( n \) such that \( v \) is live at \( n \).

Exercise 4 (Multi-Pushdown Systems). An \( n \)-dimensional multi-pushdown system \( (n\text{-MPDS}) \) is a tuple \( \mathcal{M} = (P, \Gamma, (\Delta_i)_{0 \leq i \leq n}) \) where \( n \geq 1 \) is the number of stacks, \( P \) a finite set of control states, \( \Gamma \) a finite stack alphabet, and each \( \Delta_i \subseteq P \times \Gamma \times P \times \Gamma^* \) is a finite transition relation. A configuration of an \( n \text{-MPDS} \) is a tuple \( c = (q, w_1, \ldots, w_n) \) in \( P \times (\Gamma^*)^n \). The transition relation \( \to \) on configurations is defined as \( \to = \bigcup_{0 \leq i \leq n} \gamma_i \), where

\[
(q, w_1, \ldots, A w_i, \ldots, w_n) \xrightarrow{\gamma_i} (q', w_1, \ldots, w'_i w_i, \ldots, w_n) \quad \text{iff} \quad qA \xrightarrow{\gamma'} q' w'_i w_i \in \Delta_i
\]

1. Show that the control state reachability problem, i.e. given an initial configuration \( c \in P \times \Gamma^n \) and a control state \( p \in P \), whether there exist \( w_1, \ldots, w_n \) such that \( c \xrightarrow{\ast} (p, w_1, \ldots, w_n) \) is undecidable as soon as \( n \geq 2 \).
2. Let us consider a restriction on $\rightarrow^*$: *k-bounded* runs are defined as the $k$-iterates $c \Rightarrow^k c'$ of the relation

$$ c \rightarrow c' \text{ iff } \exists i. c \rightarrow_i^* c' $$

i.e. a $k$-bounded run can be decomposed into $k$ subruns where a single PDS is running.

Show that the $k$-bounded control-state reachability problem, i.e. given an initial configuration $c$ in $P \times \Gamma^n$ and a control state $p \in P$, whether there exist $w_1, \ldots, w_n$ such that $c \Rightarrow^k (q, w_1, \ldots, w_n)$ is decidable.