Visibly Pushdown Languages

December 1, 2022

Motivation

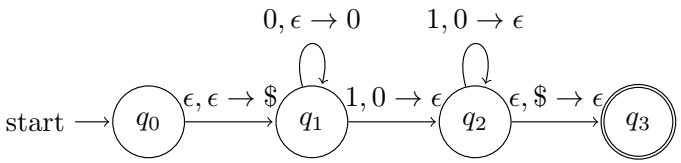
Two context-free languages

$$\{0^n 1^n \mid n \geq 0\}$$

Its grammar:

$$S \to 0S1 \mid \epsilon.$$

Its automaton:

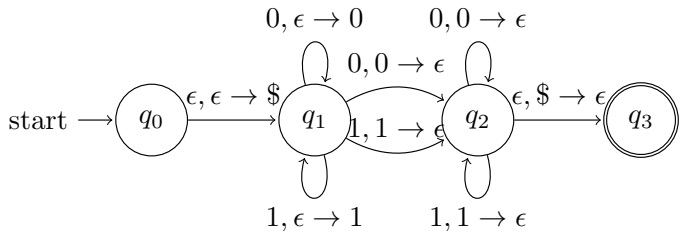


$$\{ww^R \mid w \in \{0,1\}^*\}$$

Its grammar:

$$S \to 0S0 \mid 1S1 \mid \epsilon.$$

Its automaton:



Motivation

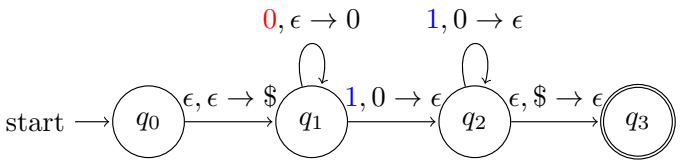
Their difference

$$\{0^n 1^n \mid n \geq 0\}$$

Its grammar:

$$S \to 0S1 \mid \epsilon.$$

Its automaton:

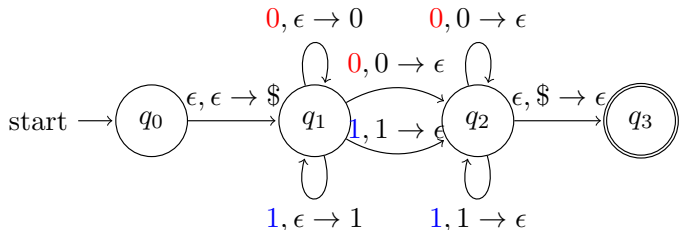


$$\{ww^R \mid w \in \{0,1\}^*\}$$

Its grammar:

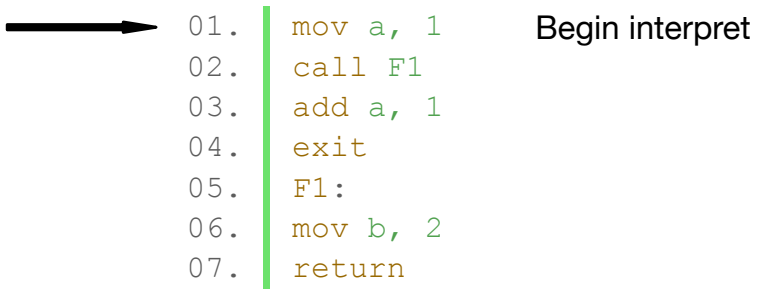
$$S \to 0S0 \mid 1S1 \mid \epsilon.$$

Its automaton:



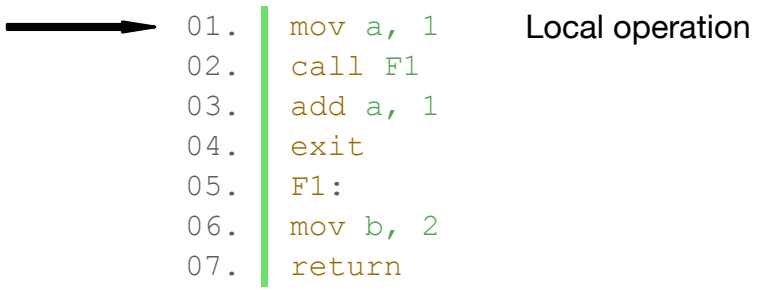
A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:



A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:



A Special Class of Context-Free Languages

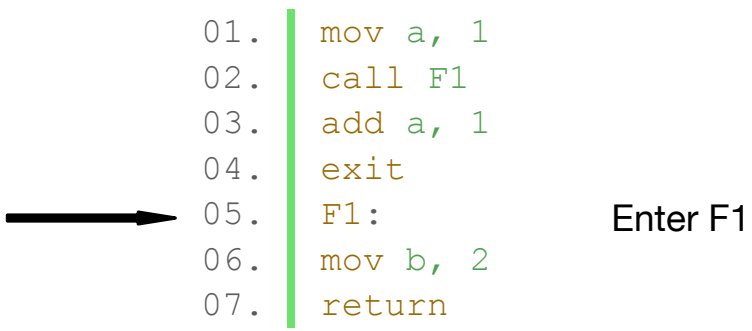
Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:


```
01. mov a, 1
02. call F1
03. add a, 1
04. exit
05. F1:
06. mov b, 2
07. return
```

Call operation

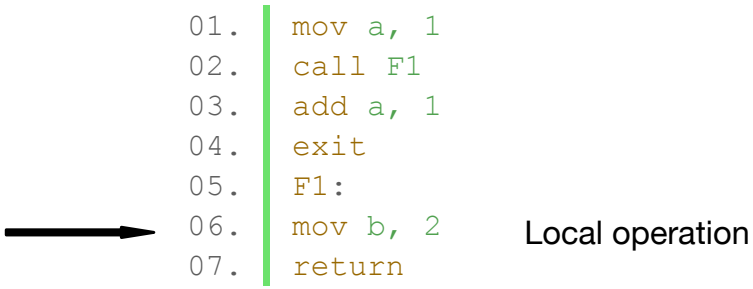
A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:



A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:



A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:

```
01. mov a, 1
02. call F1
03. add a, 1
04. exit
05. F1:
06. mov b, 2
07. return
```

→ Return operation

A Special Class of Context-Free Languages

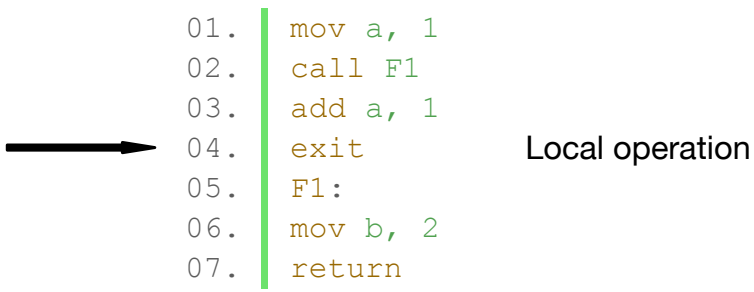
Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:


```
01. mov a, 1
02. call F1
03. add a, 1
04. exit
05. F1:
06. mov b, 2
07. return
```

Local operation

A Special Class of Context-Free Languages

Some context-free language's automaton pushes only when reading specific characters and pops only when facing other specific characters.
Leverage language interpreter as an example:



Visibly Pushdown languages

Definition via pushdown automata

1. **Pushdown alphabet:** $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. Σ_c is for call alphabet; Σ_r for return alphabet; Σ_l for local alphabet.

Visibly Pushdown languages

Definition via pushdown automata

1. **Pushdown alphabet:** $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. Σ_c is for call alphabet; Σ_r for return alphabet; Σ_l for local alphabet.
2. **(Nondeterministic) Pushdown automaton:** $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ (Based on $\widetilde{\Sigma}$).

Visibly Pushdown languages

Definition via pushdown automata

1. **Pushdown alphabet:** $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. Σ_c is for call alphabet; Σ_r for return alphabet; Σ_l for local alphabet.
2. **(Nondeterministic) Pushdown automaton:** $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ (Based on $\widetilde{\Sigma}$).
3. **States:** Q is a finite set of states; $Q_{in} \subseteq Q$ a **set** of initial states; $Q_F \subseteq Q$ a set of accepting finite states.

Visibly Pushdown languages

Definition via pushdown automata

1. **Pushdown alphabet:** $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. Σ_c is for call alphabet; Σ_r for return alphabet; Σ_l for local alphabet.
2. **(Nondeterministic) Pushdown automaton:** $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ (Based on $\widetilde{\Sigma}$).
3. **States:** Q is a finite set of states; $Q_{in} \subseteq Q$ a **set** of initial states; $Q_F \subseteq Q$ a set of accepting finite states.
4. **Stack alphabet:** Γ is a finite alphabet with a special bottom-of-stack symbol $\perp$.

Visibly Pushdown languages

Definition via pushdown automata

1. **Pushdown alphabet:** $\widetilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. Σ_c is for call alphabet; Σ_r for return alphabet; Σ_l for local alphabet.
2. **(Nondeterministic) Pushdown automaton:** $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ (Based on $\widetilde{\Sigma}$).
3. **States:** Q is a finite set of states; $Q_{in} \subseteq Q$ a **set** of initial states; $Q_F \subseteq Q$ a set of accepting finite states.
4. **Stack alphabet:** Γ is a finite alphabet with a special bottom-of-stack symbol $\perp$.
5. **Transition function:** $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma - \{\perp\}))$ $\cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_l \times Q)$.

How Visibly Pushdown Automaton Accepts Words?

For a word $w = a_1 ... a_k \in \Sigma^*$ ($\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$),
a run of M on w is a sequence $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$, where
$q_0 \in Q_{in}$, each $q_i \in Q$, each $\sigma_i \in St$ ($St = (\Gamma - \{\perp\})^*.\{\perp\}$), $\sigma_0 = \perp$,
and for every $1 \leq i \leq k$ the following holds:

1. **[Push]** If a_i is a call, then for some $\gamma \in \Gamma$, $(q_i, a_i, q_{i+1}, \gamma) \in \delta$ and $\sigma_{i+1} = \gamma.\sigma_i$.

How Visibly Pushdown Automaton Accepts Words?

For a word $w = a_1 ... a_k \in \Sigma^*$ ($\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$),
a run of M on w is a sequence $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$, where
$q_0 \in Q_{in}$, each $q_i \in Q$, each $\sigma_i \in St$ ($St = (\Gamma - \{\perp\})^*.\{\perp\}$), $\sigma_0 = \perp$,
and for every $1 \leq i \leq k$ the following holds:

1. **[Push]** If a_i is a call, then for some $\gamma \in \Gamma, (q_i, a_i, q_{i+1}, \gamma) \in \delta$ and $\sigma_{i+1} = \gamma.\sigma_i$.
2. **[Pop]** If a_i is a return, then for some $\gamma \in \Gamma, (q_i, a_i, \gamma, q_{i+1}) \in \delta$ and either $\gamma \neq \perp$ and $\sigma_i = \gamma.\sigma_{i+1}$ or $\gamma = \perp$ and $\sigma_i = \sigma_{i+1} = \perp$.

How Visibly Pushdown Automaton Accepts Words?

For a word $w = a_1 ... a_k \in \Sigma^*$ ($\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$),
a run of M on w is a sequence $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$, where
$q_0 \in Q_{in}$, each $q_i \in Q$, each $\sigma_i \in St$ ($St = (\Gamma - \{\perp\})^*.\{\perp\}$), $\sigma_0 = \perp$,
and for every $1 \leq i \leq k$ the following holds:

1. **[Push]** If a_i is a call, then for some $\gamma \in \Gamma, (q_i, a_i, q_{i+1}, \gamma) \in \delta$ and $\sigma_{i+1} = \gamma.\sigma_i$.
2. **[Pop]** If a_i is a return, then for some $\gamma \in \Gamma, (q_i, a_i, \gamma, q_{i+1}) \in \delta$ and either $\gamma \neq \perp$ and $\sigma_i = \gamma.\sigma_{i+1}$ or $\gamma = \perp$ and $\sigma_i = \sigma_{i+1} = \perp$.
3. **[Local]** If a_i is a local action, then $(q_i, a_i, q_{i+1}) \in \delta$ and $\sigma_{i+1} = \sigma_i$.

How Visibly Pushdown Automaton Accepts Words?

For a word $w = a_1 ... a_k \in \Sigma^*$ ($\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$),
a run of M on w is a sequence $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$, where
$q_0 \in Q_{in}$, each $q_i \in Q$, each $\sigma_i \in St$ ($St = (\Gamma - \{\perp\})^*.\{\perp\}$), $\sigma_0 = \perp$,
and for every $1 \leq i \leq k$ the following holds:

1. **[Push]** If a_i is a call, then for some $\gamma \in \Gamma, (q_i, a_i, q_{i+1}, \gamma) \in \delta$ and $\sigma_{i+1} = \gamma.\sigma_i$.
2. **[Pop]** If a_i is a return, then for some $\gamma \in \Gamma, (q_i, a_i, \gamma, q_{i+1}) \in \delta$ and either $\gamma \neq \perp$ and $\sigma_i = \gamma.\sigma_{i+1}$ or $\gamma = \perp$ and $\sigma_i = \sigma_{i+1} = \perp$.
3. **[Local]** If a_i is a local action, then $(q_i, a_i, q_{i+1}) \in \delta$ and $\sigma_{i+1} = \sigma_i$.

A run ρ is accepting if the last state is a final state, i.e., $q_k \in Q_F$.

More about Visibly Pushdown Automata

ϵ-transition

The above automata definition forbids any ϵ-transitions.

1. ϵ-transitions that push or pop must be forbidden.

More about Visibly Pushdown Automata

ϵ-transition

The above automata definition forbids any ϵ-transitions.

1. ϵ-transitions that push or pop must be forbidden.
2. The expressiveness will not increase if we allow ϵ-transitions that do not push or pop (Maintain a ϵ-reachability relation, similar to ϵ-*NFA* $\equiv$ *NFA*).

More about Visibly Pushdown Automata

Q_{in} is a state set, not a state

$M_{Q_{in}} \equiv M_{q_0}$

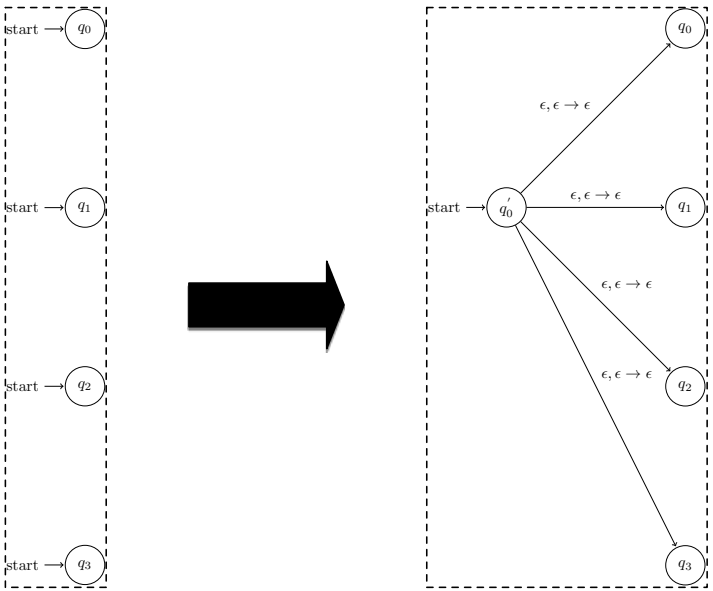
1. For a $M = (Q, q_0, \Gamma, \delta, Q_F)$, we can build an equivalent $M' = (Q, \{q_0\}, \Gamma, \delta, Q_F)$.

More about Visibly Pushdown Automata

Q_{in} is a state set, not a state

$M_{Q_{in}} \equiv M_{q_0}$

1. For a $M = (Q, q_0, \Gamma, \delta, Q_F)$, we can build an equivalent $M' = (Q, \{q_0\}, \Gamma, \delta, Q_F)$.
2. For a t $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$, we can build an equivalent $M' = (Q, q_0', \Gamma, \delta, Q_F)$ (ϵ-transitions that do not push or pop).



More about Visibly Pushdown languages

$\widetilde{\Sigma}$ does matter

Let $\widetilde{\Sigma} = \{\{0\}, \{1\}, \{\}\}$ (call,return,local)

$A = \{0^n 1^n \mid n \geq 1\}$ is VPL.

$B = \{1^n 0^n \mid n \geq 1\}$ is not VPL.

Overview of Closure properties

Language	**Union**	**Intersection**	**Complement**	**Concat.**	*****
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DFL	No	No	Yes	No	No
VPL[1]	Yes	Yes	Yes	Yes	Yes

Table: Clousure Properties

[1] L_1 and L_2 should use same $\widetilde{\Sigma}$

Union

Given L_1 and L_2 accepted by $M_1 = (Q_1, Q_{in1}, \Gamma, \delta_1, Q_{F1})$ and $M_1 = (Q_2, Q_{in2}, \Gamma, \delta_2, Q_{F2})$, $L_1 \cup L_2$ is accepted by $M' =$

$$(Q_1 \cup Q_2, Q_{in1} \cup Q_{in2}, \Gamma, \delta_1 \cup \delta_2, Q_{F1} \cup Q_{F2})$$

Intersection

Given L_1 and L_2 accepted by $M_1 = (Q_1, Q_{in1}, \Gamma, \delta_1, Q_{F1})$ and $M_2 = (Q_2, Q_{in2}, \Gamma, \delta_2, Q_{F2})$ is accepted by $M' =$

$$(Q_1 \times Q_2, Q_{in1} \times Q_{in2}, \Gamma \times \Gamma, \delta', Q_{F1} \times Q_{F2})$$

. δ' consists of

1. **[Local]** $\{(q_{1a} \times q_{2a}, a, q_{1b} \times q_{2b}) \mid (q_{ia}, a, q_{ib}) \in \delta_i\}$
2. **[Push]** $\{(q_{1a} \times q_{2a}, a, q_{1b} \times q_{2b}, \gamma_1 \times \gamma_2) \mid (q_{1i}, a, q_{1i}, \gamma_i) \in \delta_i\}$
3. **[Pop]** $\{(q_{1a} \times q_{2a}, a, \gamma_1 \times \gamma_2, q_{1b} \times q_{2b}) \mid (q_{ia}, a, \gamma_i, q_{ib}) \in \delta_i\}$

Concatenation

Intuition: Nondeterministically guessing a split of the input word w into w_1 and w_2.

1. Link M_1's accepting states and M_2's initial states with ϵ-transition with no push and pop.
2. Modify $M_1 = (Q_1, Q_{in1}, \Gamma, \delta_1, Q_{F1})$ to $(Q_1, Q_{in1}, \Gamma \times \{a\}, \delta_1', Q_{F1})$;
   Modify $M_2 = (Q_2, Q_{in2}, \Gamma, \delta_1, Q_{F2})$ to $(Q_1, Q_{in2}, \Gamma \times \{b\}, \delta_2', Q_{F2})$;
   When simulating M_2, the stack-alphabet for M_1 is treated as bottom-of-stack.

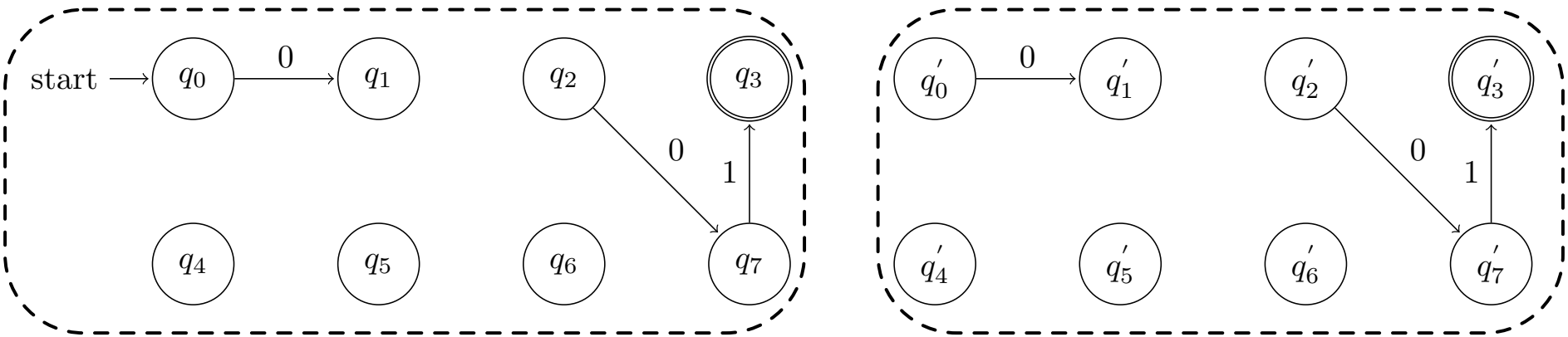
*

Let $M = (Q, Q_{in}, \Gamma, Q_F)$ be a VPA accepting L. Build a M^* to accept L^*.

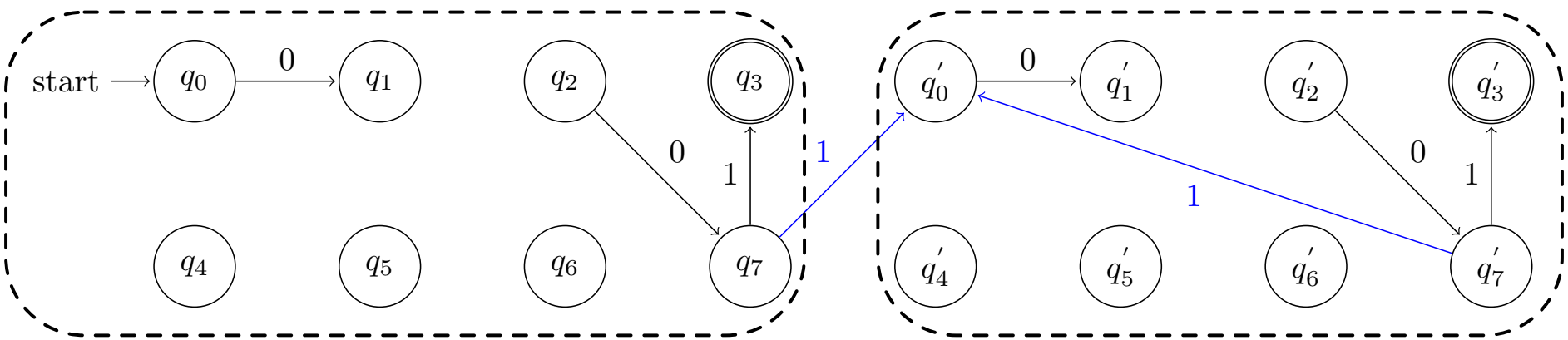
Intuition: M^* simulates M step by step, but when M changes its state to a final state, M^* can nondeterministically update its state to an initial state and restart M.

Difficulty: After restarting, M^* must treat the stack as if it is empty.

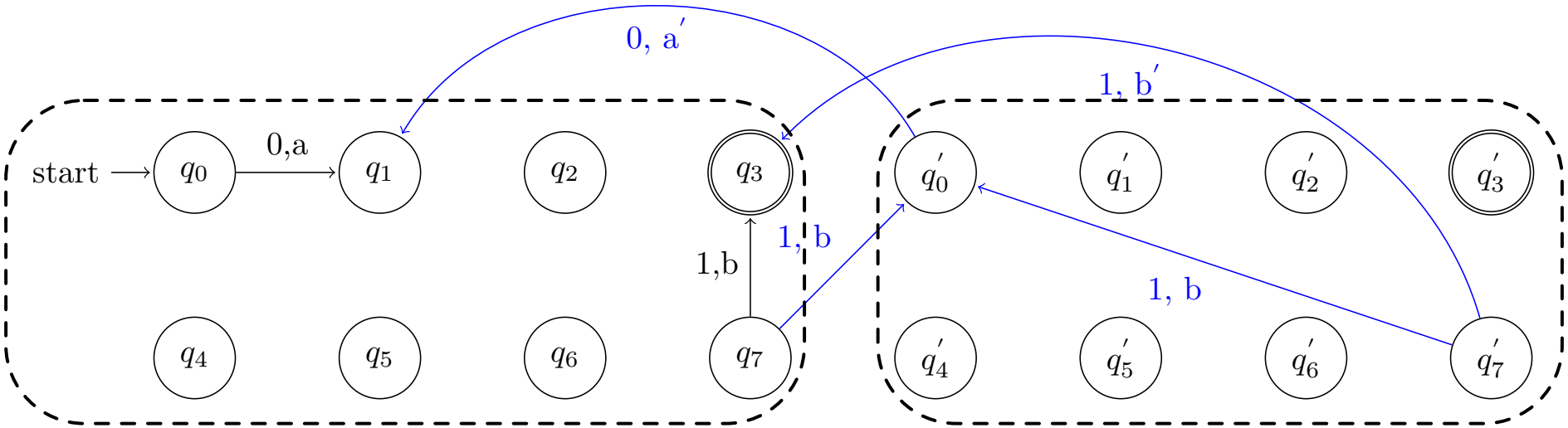
For each local transition $(q, a, p) \in \delta$, (1) M^* contains the transitions (q, a, p) and (q', a, p'), and (2) if $p \in Q_F$, then the transitions (q, a, r') and (q', a, r') for each $r \in Q_{in}$.



For each local transition $(q, a, p) \in \delta$, (1) M^* contains the transitions (q, a, p) and (q', a, p'), and (2) if $p \in Q_F$, then the transitions (q, a, r') and (q', a, r') for each $r \in Q_{in}$.



For each push transition $(q, a, p, \gamma) \in \delta$, (1) M^* contains the transitions (q, a, p, γ) and (q', a, p, γ'), and (2) if $p \in Q_F$, then the transitions (q, a, r', γ) and (q', a, r', γ) for each $r \in Q_{in}$.



Push Example

$M^* = (M_{old}, M_{new})$ on $w = w_1 ... w_n$

1. M_{old} on $w_1 ... w_i$. The stack's content is $(b, a, \perp)$
2. M_{new} on $w_{i+1} ... w_j$. The stack's content is $(c', b, a, \perp)$
3. M_{old} on $w_{j+1} ... w_n$. The stack's content is $(..., c', b, a, \perp)$

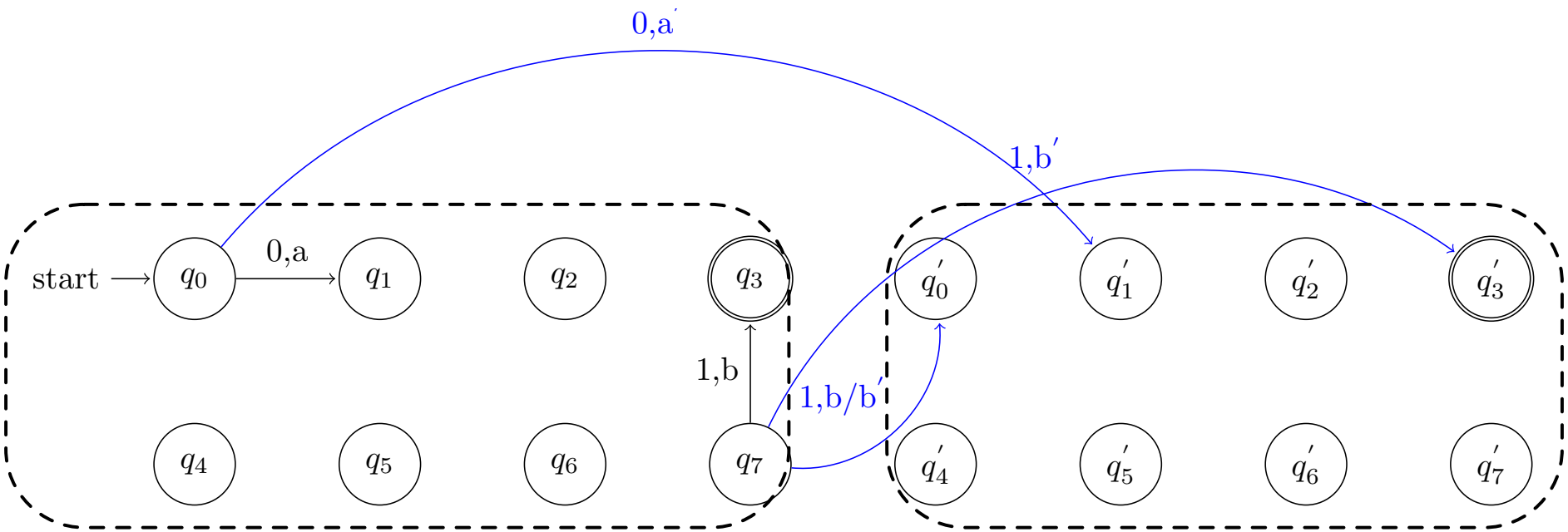
Push Example

$M^* = (M_{old}, M_{new})$ on $w = w_1 ... w_n$

1. M_{old} on $w_1 ... w_i$. The stack's content is $(b, a, \perp)$
2. M_{new} on $w_{i+1} ... w_j$. The stack's content is $(c', b, a, \perp)$
3. M_{old} on $w_{j+1} ... w_n$. The stack's content is $(..., c', b, a, \perp)$
4. When and how to pop c' and the content below it?

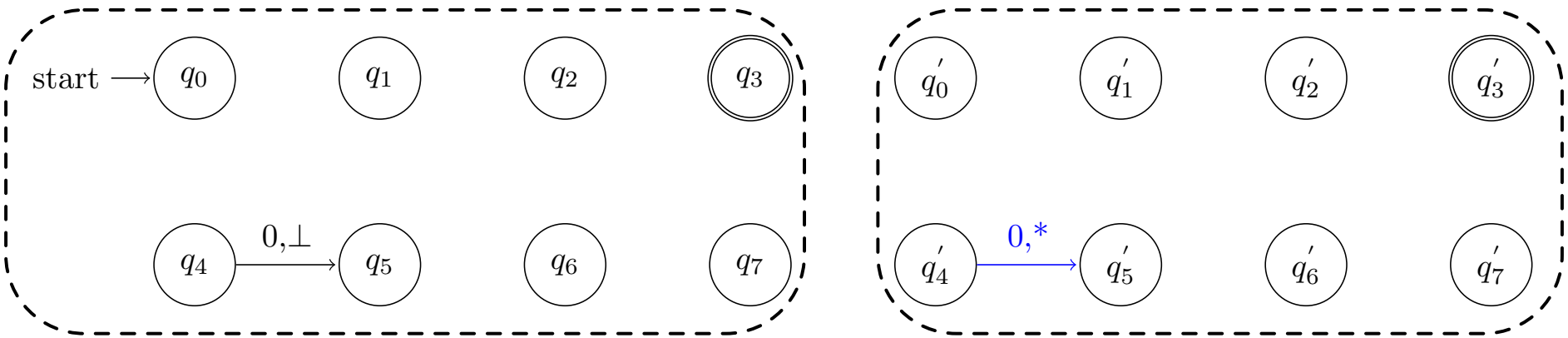
Pop (1)

For each pop transition $(q, a, \gamma, p) \in \delta$, (1) M^* contains the transitions (q, a, γ, p) and (q, a, γ', p'), and (2) if $p \in Q_F$, then the transitions (q, a, γ, r') and (q', a, γ', r') for each $r \in Q_{in}$.



Pop (2)

For each pop transition $(q, a, \perp, p) \in \delta$, (1) M^* contains the transitions (q', a, γ, p') for each $\gamma \in \Gamma \cup \Gamma'$, and (2) if $p \in Q_F$, then the transitions (q', a, γ, r') and $r \in Q_{in}$.



Determinization

For any VPA M over $\widetilde{\Sigma}$, there is a deterministic VPA M' over $\widetilde{\Sigma}$ such that $L(M') = L(M)$.

Determinization

Where non-determinization comes from?

1. Q_{in} may contain multiple states.
2. For each $q \in Q$ and $a \in \Sigma_I$, δ may contain multiple (q, a, q').

Determinization

Where non-determinization comes from?

1. Q_{in} may contain multiple states.
2. For each $q \in Q$ and $a \in \Sigma_l$, δ may contain multiple (q, a, q').
3. For each $q \in Q$ and $a \in \Sigma_c$, δ may contain multiple (q, a, q', γ).
4. For each $q, \in Q$, $a \in \Sigma_r$, and $\gamma \in \Gamma$, δ may contain multiple (q, a, γ, q').

Determinization

Intuition

Assume that there is no *local* operations between *call* and *return*.

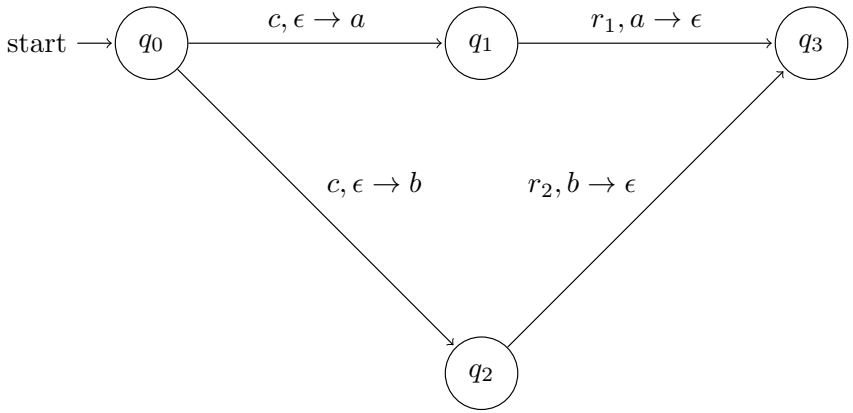


Figure: Partial automaton

$$c \in \Sigma_c$$

$$\{a, b\} \subseteq \Gamma$$

$$\{r_1, r_2\} \subseteq \Sigma_r$$

We can search out which push operations are matched by each pop.
For (q_1, r_1, a, q_3), we can search
$\{d \mid (q, d, q_1, a) \in \delta\}$=$\{c\}$.
For (q_2, r_2, a, q_3), we can search
$\{d \mid (q, d, q_2, a) \in \delta\}$=$\{c\}$.

Determinization

Intuition

Assume that there is no *local* operations between *call* and *return*.

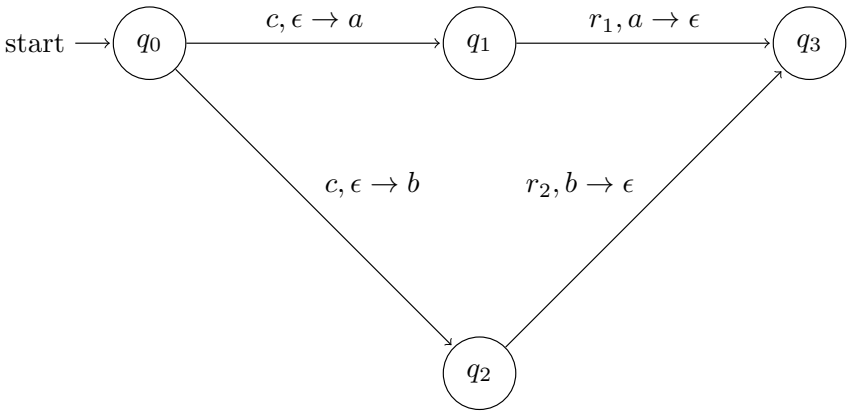


Figure: Partial automaton

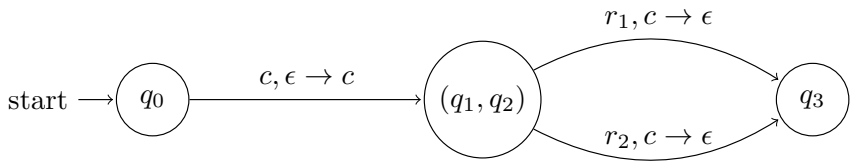


Figure: Deterministic automaton

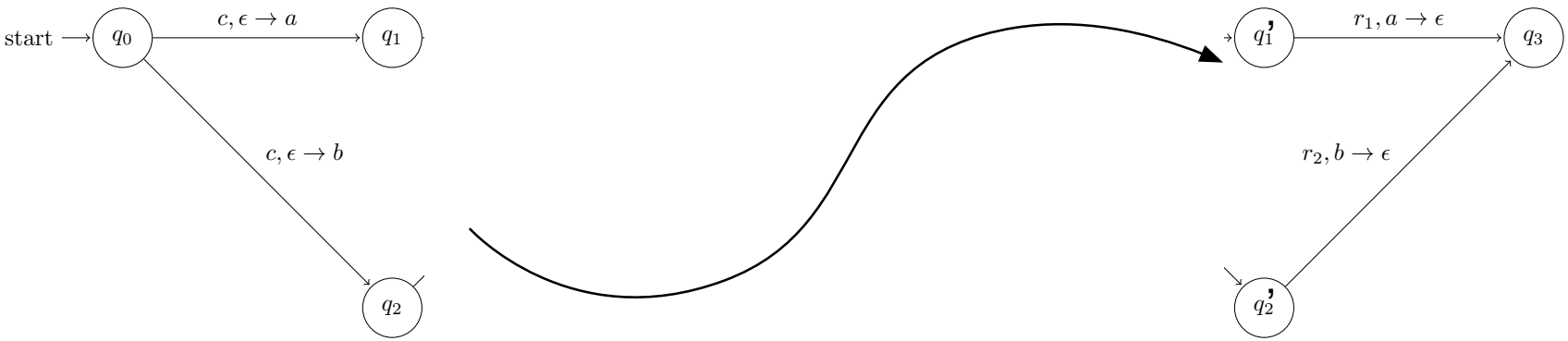
$$c \in \Sigma_c$$

$$\{a, b\} \subseteq \Gamma$$

$$\{r_1, r_2\} \subseteq \Sigma_r$$

Determinization

Problem



We don't know whether they are linked

Determinization

Construction

Let L be accepted by a VPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$. We construct an equivalent deterministic VPA $M' = (Q', Q'_{in}, \Gamma', \delta', Q'_F)$.

1. Let $Q' = 2^{Q \times Q} \times 2^{Q}$.

Determinization

Construction

Let L be accepted by a VPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$. We construct an equivalent deterministic VPA $M' = (Q', Q'_{in}, \Gamma', \delta', Q'_F)$.

1. Let $Q' = 2^{Q \times Q} \times 2^{Q}$.
2. Let Id_Q denotes the set $\{(q, q) \mid q \in Q\}$, then $Q'_{in} = \{Id_Q, Q_{in}\}$.

Determinization

Construction

Let L be accepted by a VPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$. We construct an equivalent deterministic VPA $M' = (Q', Q'_{in}, \Gamma', \delta', Q'_F)$.

1. Let $Q' = 2^{Q \times Q} \times 2^{Q}$.
2. Let Id_Q denotes the set $\{(q, q) \mid q \in Q\}$, then $Q'_{in} = \{Id_Q, Q_{in}\}$.
3. The stack alphabet Γ' is the set of elements (S, R, a), where $(S, R) \in Q'$ and $a \in \Gamma_c$.

Determinization

Construction

Let L be accepted by a VPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$. We construct an equivalent deterministic VPA $M' = (Q', Q'_{in}, \Gamma', \delta', Q'_F)$.

1. Let $Q' = 2^{Q \times Q} \times 2^{Q}$.
2. Let Id_Q denotes the set $\{(q, q) \mid q \in Q\}$, then $Q'_{in} = \{Id_Q, Q_{in}\}$.
3. The stack alphabet Γ' is the set of elements (S, R, a), where $(S, R) \in Q'$ and $a \in \Gamma_c$.

We run a fix-point algorithm to get an invariant, i.e., the transition relation, including the stack elements and control states.

Determinization

Local

For every $a \in \Gamma_l$, $((S, R), a, (S', R')) \in \delta'$ where

1. $R' = \{q' \mid \exists q \in R : (q, a, q') \in \delta\}$
2. $S' = \{(q, q') \mid \exists q'' : (q, q'') \in S, (q'', a, q') \in \delta\}$

Determinization

Call

For every $a \in \Gamma_c$, $((S, R), a, (Id_Q, R'), (S, R, a)) \in \delta'$ where

1. $R' = \{q' \mid \exists q \in R, \gamma \in \Gamma : (q, a, q', \gamma) \in \delta\}$

Determinization

Return 1

For every $a \in \Gamma_r$, $((S, R), a, (S', R', a'), (S'', R'')) \in \delta'$ if (S'', R'') satisfies the following:
Let $Update = \{(q, q') \mid \exists q_1, q_2 \in Q, \gamma \in \Gamma : (q, a', q_1, \gamma) \in \delta, (q_1, q_2) \in S, (q_2, a, \gamma, q') \in \delta\}$.

1. $R'' = \{q' \mid \exists q \in R', (q, q') \in Update\}$
2. $S'' = \{(q, q') \mid \exists q_3 \in R : (q, q_3) \in S', (q_3, q') \in Update\}$

Determinization

Return 2

For every $a \in \Gamma_r$, $((S, R), a, \perp, (S', R')) \in \delta'$ if

1. $R' = \{q' \mid \exists q \in R : (q, a, \perp, q') \in \delta\}$
2. $S' = \{(q, q') \mid \exists q'' : (q, q'') \in S, (q'', a, \perp, q') \in \delta\}$

The set of final state is $Q'_F = \{(S, R) \mid R \cap Q_F \neq \emptyset\}$.

Corollary

The class of visibly pushdown languages is closed under complementation.

Visibly Pushdown Grammar

A context-free grammar $G = (V, S, P)$ over Σ is a visibly pushdown grammar with respect to the partitioning $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$, if the set V of variables is partitioned into two disjoint sets V^0 and V^1, such that all the productions in P are of one the following forms:

1. $X \to \epsilon$
2. $X \to aY$ such that $X \in V^0 \Rightarrow a \in \Sigma_l \wedge Y \in V^0$
3. $X \to aYbZ$ such that $a \in \Sigma_c \wedge b \in \Sigma_r \wedge Y \in V^0$ and $X \in V^0 \Rightarrow Z \in V^0$

The variables in V^0 derive only well-matched words. The variables in V^1 derive words that can contain unmatched calls as well as unmatched returns.

Grammar to Automaton

Let $G = (V, S, P)$ be a visibly pushdown grammar. We build a VPA M_G that accepts $L(G)$ as follows.

1. The set of states of M_G is V.
2. The unique initial state is S.
3. The stack alphabet is $V \times \Sigma_r$ along with $\perp$ and a dummy symbol \$.

Grammar to Automaton

Transition

1. **Local** $a \in \Sigma_l$: δ contains (X, a, Y) for each $X \to aY$.

Grammar to Automaton

Transition

1. **Local** $a \in \Sigma_l$: δ contains (X, a, Y) for each $X \to aY$.
2. **Push** $a \in \Sigma_c$: δ contains $(X, a, Y, \$)$ for each $X \to aY$; and $(X, a, Y, (b, Z))$ for each $X \to aYbZ$.

Grammar to Automaton

Transition

1. **Local** $a \in \Sigma_l$: δ contains (X, a, Y) for each $X \to aY$.
2. **Push** $a \in \Sigma_c$: δ contains $(X, a, Y, \$)$ for each $X \to aY$; and $(X, a, Y, (b, Z))$ for each $X \to aYbZ$.
3. **Pop** $a \in \Sigma_r$: δ contains $(X, a, \bot, Y)$ and $(X, a, \$, Y)$ for each $X \to aY$; and if X is a nullable symbol and is in V^0, then for each variable Y and symbol a, δ contains $(X, a, (a, Y), Y)$.

Grammar to Automaton

Transition

1. **Local** $a \in \Sigma_l$: δ contains (X, a, Y) for each $X \to aY$.
2. **Push** $a \in \Sigma_c$: δ contains $(X, a, Y, \$)$ for each $X \to aY$; and $(X, a, Y, (b, Z))$ for each $X \to aYbZ$.
3. **Pop** $a \in \Sigma_r$: δ contains $(X, a, \bot, Y)$ and $(X, a, \$, Y)$ for each $X \to aY$; and if X is a nullable symbol and is in V^0, then for each variable Y and symbol a, δ contains $(X, a, (a, Y), Y)$.

The accepting condition for M_A is that the state should be nullable, and the top of the stack should be either $\bot$ or \$ (which means that there is no requirement concerning matching return).

Automaton to Grammar

Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$. We construct a visibly pushdown grammar G_M that generates $L(M)$.

1. For each $q \in Q$, the set V^1 has two variables X_q and Y_q.
2. For every pair of states q, p and stack symbol γ, the set V^0 has a variable $Z_{q,\gamma,p}$.
3. The start variables are X_q for each $q \in Q_{in}$.

Automaton to Grammar

Productions

1. For each state q, if $q \in Q_F$, add $X_q \to \epsilon$ and $Y_q \to \epsilon$.
2. For each state p and each stack symbol γ, add $Z_{p,\gamma,p} \to \epsilon$.
3. For each local a and states q, p, if δ contains (q, a, p), add $X_q \to aX_p$ and $Y_q \to aY_p$.
4. For each local a and states p, q, q', and stack symbol γ, if δ contains (q, a, p), add $Z_{q,\gamma,q'} \to aZ_{p,\gamma,q'}$.

Automaton to Grammar

Productions

1. For each state q, if $q \in Q_F$, add $X_q \to \epsilon$ and $Y_q \to \epsilon$.
2. For each state p and each stack symbol γ, add $Z_{p,\gamma,p} \to \epsilon$.
3. For each local a and states q, p, if δ contains (q, a, p), add $X_q \to aX_p$ and $Y_q \to aY_p$.
4. For each local a and states p, q, q', and stack symbol γ, if δ contains (q, a, p), add $Z_{q,\gamma,q'} \to aZ_{p,\gamma,q'}$.
5. For each call a, return b, stack symbol γ,and states q, q', p, p', if δ contains (q, a, γ, p) and (q', b, p', γ), add $X_q \to aZ_{p,\gamma,q'}bX_{p'}$ and $Y_q \to aZ_{p,\gamma,q'}bY_{p'}$
6. For each call a, return b, stack symbol γ,and states q, q', p, p', γ, if δ contains (q, a, γ, p) and (q', b, p', γ), add $Z_{q,\gamma',r} \to aZ_{p,\gamma,q'}bZ_{p',\gamma',r}$.
7. For each call a, states q, p, and stack symbol γ, If δ contains (q, a, γ, p), add $X_q \to aY_p$ and $Y_q \to aY_p$.

Automaton to Grammar

Productions

1. For each state q, if $q \in Q_F$, add $X_q \to \epsilon$ and $Y_q \to \epsilon$.
2. For each state p and each stack symbol γ, add $Z_{p,\gamma,p} \to \epsilon$.
3. For each local a and states q, p, if δ contains (q, a, p), add $X_q \to aX_p$ and $Y_q \to aY_p$.
4. For each local a and states p, q, q', and stack symbol γ, if δ contains (q, a, p), add $Z_{q,\gamma,q'} \to aZ_{p,\gamma,q'}$.
5. For each call a, return b, stack symbol γ,and states q, q', p, p', if δ contains (q, a, γ, p) and (q', b, p', γ), add $X_q \to aZ_{p,\gamma,q'}bX_{p'}$ and $Y_q \to aZ_{p,\gamma,q'}bY_{p'}$
6. For each call a, return b, stack symbol γ,and states q, q', p, p', γ, if δ contains (q, a, γ, p) and (q', b, p', γ), add $Z_{q,\gamma',r} \to aZ_{p,\gamma,q'}bZ_{p',\gamma',r}$.
7. For each call a, states q, p, and stack symbol γ, If δ contains (q, a, γ, p), add $X_q \to aY_p$ and $Y_q \to aY_p$.
8. For each return a, states q, p, if δ contains $(q, a, \perp, p)$, add $X_q \to aX_p$.

Content not Covered

1. Relation to regular tree languages
2. Relation to Monadic second-order logic
3. Decision problems