Alternative deep pushdown automata and their applications

Bc. Radovan Klembara

Faculty of information technology, Brno University of Technology

Deep pushdown automata

The deep pushdown automata, their definition and strength were presented in Acta Informatica by professor Meduna [1]. The Figure 1 shows usage of one expansion rule. The rule is $2sB \to sbBA$.

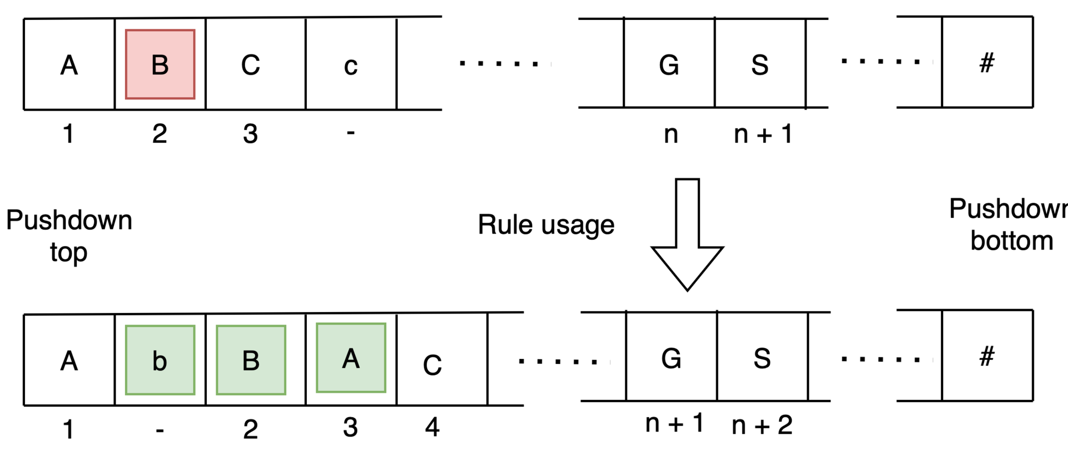

Figure 1. Pushdown before and after usage of a rule.

Example

Demonstration of deep pushdown automaton accepting word $w = aabbcc$, from $L = \{a^n b^n c^n; n \geq 1\}$.

${}_2M = (\{s,q,p\}, \{a,b,c\}, \{A,S,\#\}, R, s, S, \{f\})$, where R are rules:

1. $1sS \to qAA$
2. $1qA \to paAb$
3. $1qA \to fab$
4. $2pA \to qAc$
5. $1fA \to fc$

Acceptance of word w can look like this:

$$
\begin{aligned}
(s, aabbcc, S\#) \;& {}_e\Rightarrow (q, aabbcc, AA\#) && [1]\\
& {}_e\Rightarrow (p, aabbcc, aAbA\#) && [2]\\
& {}_p\Rightarrow (p, abbcc, AbA\#) && \\
& {}_e\Rightarrow (q, abbcc, AbAc\#) && [4]\\
& {}_e\Rightarrow (f, abbcc, abbAc\#) && [3]\\
& {}_p\Rightarrow^* (f, cc, Ac\#) && \\
& {}_e\Rightarrow (f, cc, cc\#) && [5]\\
& {}_p\Rightarrow^* (f, \epsilon, \#) &&
\end{aligned}
$$

Strength

The strength of PD_n with depth equal to *1* is ${}_{deep}\mathrm{PD}_1 = \mathbf{CF}$. PD_n creatis infinite hierarchy of language families. Resulting in $\mathbf{CF} = {}_{deep}\mathrm{PD}_1 \subset {}_{deep}\mathrm{PD}_2 \subset \dots \subset {}_{deep}\mathrm{PD}_n \subset {}_{deep}\mathrm{PD}_\infty = \mathbf{CS}$.

Parallel deep pushdown automata

Introduces parallelism to basic deep pushdown automata. The Figure 1 shows usage of one expansion rule. The rule is $s(A,B) \to s(A,bB)$.

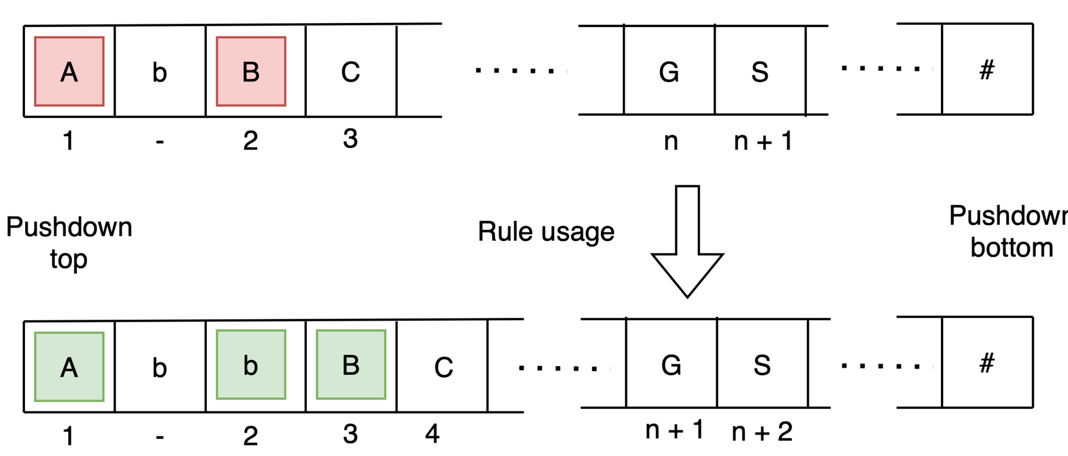

Figure 2. Pushdown before and after usage of a parallel rule.

Definition

Parallel deep pushdown automaton (PPD_n) is a septuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, defined same as pushdown automaton expcept for rules R:

- R is set of rules, in form $q(A_1, A_2, \dots, A_k) \to p(\alpha_1, \alpha_2, \dots, \alpha_k)$,

Example

Demonstration of PPD_n accepting word $w = aaabbcc$, from language $L = \{a^i b^j c^k; \forall i,j,k \in \mathbb{N}_0; i \geq j \geq k\}$.

${}_2M = (\{s\}, \{a,b,c\}, \{a,b,c,S,A,B,\#\}, R, s, S, \{s\})$, where R are rules:

1. $s(S) \to s(\epsilon)$
2. $s(S) \to s(ABc)$
3. $s(A,B) \to s(a,b)$
4. $s(A,B) \to s(aA,bB)$
5. $s(A,B) \to s(aA,bBc)$
6. $s(A) \to s(aA)$

Acceptance of word w can look like this:

$$
\begin{aligned}
(s, aaabbcc, S\#) \;& {}_e\Rightarrow (s, aaabbcc, ABc\#) && [2]\\
& {}_e\Rightarrow (s, aaabbcc, aAbBcc\#) && [5]\\
& {}_p\Rightarrow (s, aabbcc, AbBcc\#) && \\
& {}_e\Rightarrow (s, aaabbcc, aAbBcc\#) && [6]\\
& {}_p\Rightarrow (s, abbcc, AbBcc\#) && \\
& {}_e\Rightarrow (s, abbcc, abbcc\#) && [3]\\
& {}_p\Rightarrow^* (s, \epsilon, \#) &&
\end{aligned}
$$

Simulation of parallel automaton by basic automaton

For each parallel rule of size *k* create *k* sub-rules. Use them in order depending on depth, where deeper access means sooner usage. To create PD_n simulating PPD_n use Algorithm 1.

Algorithm 1: Conversion of PPD_n to PD_n.

Input: $PPD_n\ {}_nM = (Q, \Sigma, \Gamma, R_1, s, S, F)$
Output: $PD_n\ {}_nN = (Q, \Sigma, \Gamma, R_2, s, S, F)$

```
R_2 = {r = apa → p; r ∈ R_1}
R_tmp = R_1 \ R_2
while R_tmp ≠ ∅ do
    take p(A_1, ..., A_k) → q(α_1, ..., α_k) from R_tmp to r
    R_2 ∪ {iqA_i → qα_i; ∀i ∈ ℕ; 1 ≤ i < k ∧ p, q, A_i, α_i ∈ r}
    R_2 ∪ {kpA_k → qα_k; p, q, A_k, α_k ∈ r}
end
return _nN = (Q, Σ, Γ, R_2, s, S, F)
```

Simulation of basic automaton by parallel automaton

The PPD_n can simulate PD_n by replacing non-input symbols with themselves exept for symbol specified by PD_n. To create PPD_n that simulating PD_n use Algorithm 2.

Algorithm 2: Conversion of PD_n to PPD_n

Input: $PD_n\ {}_nM = (Q, \Sigma, \Gamma, R_1, s, S, F)$
Output: $PPD_n\ {}_nN = (Q, \Sigma, \Gamma, R_2, s, S, F)$

```
R_2 = {r = apa → p; r ∈ R_1}
R_2 = R_2 ∪ {p(A_1, ..., A_k) → q(α_1, ..., α_k); r = kpA_k →
    qα_k ∈ R_1, ∀i ∈ ℕ; ∀A_i ∈ Γ; i < k, α_i = A_i}
return _nN = (Q, Σ, Γ, R_2, s, S, F)
```

Strength

For every PD_n it is possible to create PPD_n that accepts same language and vice versa. This results in ${}_{deep}\mathrm{PPD}_n = {}_{deep}\mathrm{PD}_n$.

Tuple parallel deep pushdown automata

Tuple parallel deep pushdown automaton has constant number of expansions per rule equal to its depth. Instead of pushdown symbol it has starting pushdown string. The Figure 3 shows usage of one expansion rule. The rule is $s(A,B,B,G) \to s(A,AC,B,\epsilon)$.

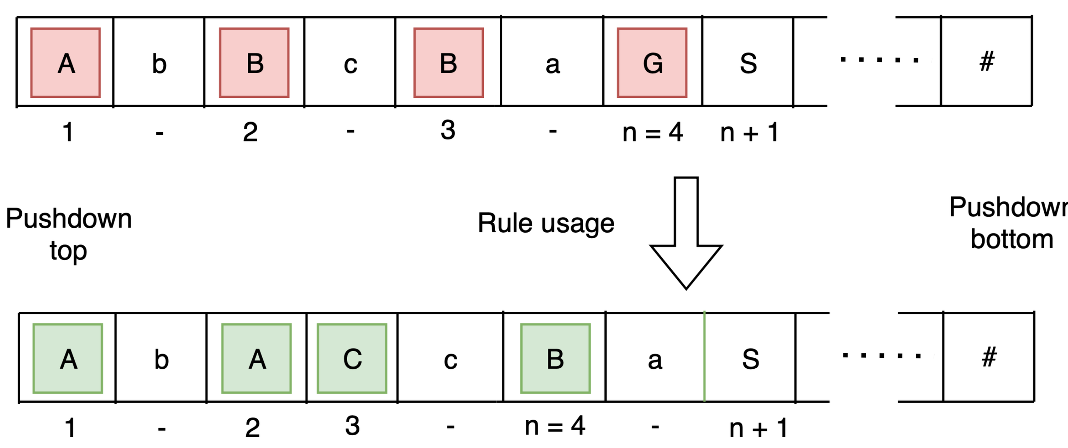

Figure 3. Pushdown before and after usage of a tuple parallel rule.

Definition

Tuple parallel deep pushdown automaton ($TPPD_n$) is a septuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where

- Q is a finite set of states,
- Σ is an input alphabet,
- Γ is a pushdown alphabet,
- R is set of rules in form $q(A_1, A_2, \dots, A_n) \to p(\alpha_1, \alpha_2, \dots, \alpha_n)$
- s is start state, $s \in Q$,
- Ω is the start pushdown string, $\Omega \in ((\Gamma \setminus \Sigma) \setminus \#)^i, i \in \mathbb{N}^+, i \geq n$,
- F is a set of final states, $F \subseteq Q$.

Example

Demonstration of $TPPD_n$ accepting word $w = aaabbbccc$ from language $L = \{a^n b^n c^n; \forall n \in \mathbb{N}^+\}$.

${}_3M = (\{s\}, \{a,b,c\}, \{a,b,c,S,\#\}, R, s, \Omega, \{s\})$, where $\Omega = SSS$ and R are rules:

1. $s(S,S,S) \to s(a,b,c)$
2. $s(S,S,S) \to s(aS,Sb,Sc)$

Acceptance of word w can look like this:

$$
\begin{aligned}
(s, aaabbbccc, SSS\#) \;& {}_e\Rightarrow (s, aaabbcc, aSSbSc\#) && [2]\\
& {}_p\Rightarrow (s, aabbbccc, SSbSc\#) && \\
& {}_e\Rightarrow (s, aabbbccc, aSSbbScc\#) && [2]\\
& {}_p\Rightarrow (s, abbbccc, SSbbScc\#) && \\
& {}_e\Rightarrow (s, abbbccc, abbbccc\#) && [1]\\
& {}_p\Rightarrow^* (s, \epsilon, \#) &&
\end{aligned}
$$

Simulation of tuple parallel automaton by basic automaton

The PD_n simulates $TPPD_n$ by creating n sub-rules in the same way as for PPD_n. $TPPD_n$ works with n non-input symbols, therefore the basic automaton has to expand its starting symbol to starting string. To create PD_n simulating $TPPD_n$ use Algorithm 3.

Algorithm 3: Conversion of $TPPD_n$ to PD_n

Input: $TPPD_n\ {}_nM = (Q, \Sigma, \Gamma_1, R_1, s, \Omega, F)$
Output: $PD_n\ {}_nN = (Q, \Sigma, \Gamma_2, R_2, s, S, F)$

```
Γ_2 = Γ_1 ∪ {S; S ∉ Γ_1}
R_2 = {1sS → sΩ}
R_2 ∪ {r = apa → p; r ∈ R_1}
R_tmp = R_1 \ R_2
while R_tmp ≠ ∅ do
    take p(A_1, ..., A_n) → q(α_1, ..., α_n) from R_tmp to r
    R_2 ∪ {kqA_k → qα_k; ∀k ∈ ℕ; 1 ≤ k < n ∧ q, A_k, α_k ∈ r}
    R_2 ∪ {npA_n → qα_n; p, q, A_n, α_n ∈ r}
end
return _nN = (Q, Σ, Γ_2, R_2, s, S, F)
```

Simulation of basic automaton by tuple parallel automaton

The parallel rules expand each non-input symbol to itself, except for symbol reffered by PD_n. $TPPD_n$ creates Ω with n "filling" symbols. To create $TPPD_n$ simulating PD_n use Algorithm 4.

Algorithm 4: Conversion of PD_n to $TPPD_n$

Input: $PD_n\ {}_nM = (Q, \Sigma, \Gamma_1, R_1, s, S_1, F)$
Output: $TPPD_n\ {}_nN = (Q, \Sigma, \Gamma_2, R_2, s, \Omega, F)$

```
Γ_2 = Γ_1 ∪ {X; X ∉ Γ_1}
Ω = S_1{X}^n
R_2 = {r = apa → p; r ∈ R_1}
R_2 ∪ {p(A_1, ..., A_k, ..., A_n) → q(α_1, ..., α_k ..., α_n);
    r = kpA_k → qα_k ∈ R_1, ∀i ∈ ℕ, ∀A_i ∈ Γ_2; i ≤ n, i ≠
    k, α_i = A_i}
R_2 ∪ {p(X_1, ..., X_n) → p(ϵ_1, ..., ϵ_n); p ∈ Q, X_i ∈ Γ_2, 0 <
    i ≤ n}
return _nN = (Q, Σ, Γ_2, R_2, s, Ω, F)
```

Strength

For every PD_n it is possible to create $TPPD_n$ that accepts same language and vice versa. This results in ${}_{deep}\mathrm{TPPD}_n = {}_{deep}\mathrm{PD}_n$.

Aplications of alternative versions

The possible usage can be in bioinformatics. Especially in searching for DNA repetitions or RNA pseudoknots. For example let L be a language $L = \{fvmvl; f,m,l,v \in \{a,g,c,t\}^*, |v| > 1\}$. This language represents DNA repetitions and PPD_n modeling it is:

${}_5M = (\{s,o,f\}, \{a,g,c,t\}, \{a,g,c,t,S,B,\#\}, R, s, S, \{f\})$, where R are rules:

1. $s(S) \to s(SSSSS)$
2. $s(S) \to s(aS)|s(gS)|s(cS)|s(tS)$
3. $s(S,S,S) \to s(S,S,aS)|s(S,S,gS)|s(S,S,cS)|s(S,S,tS)$
4. $s(S,S,S,S) \to$ $o(S,aS,S,aS)|o(S,gS,S,gS)|o(S,cS,S,cS)|o(S,tS,S,tS)$
5. $o(S,S,S,S) \to$ $f(S,aS,S,aS)|f(S,gS,S,gS)|f(S,cS,S,cS)|f(S,tS,S,tS)$
6. $f(S,S,S,S) \to$ $f(S,aS,S,aS)|f(S,gS,S,gS)|f(S,cS,S,cS)|f(S,tS,S,tS)$
7. $f(S,S,S,S,S) \to$ $f(S,S,S,S,aS)|f(S,S,S,S,gS)|f(S,S,S,S,cS)|f(S,S,S,S,tS)|$ $f(\epsilon,\epsilon,\epsilon,\epsilon,\epsilon)$

Acceptance of the word $w = agtccttgtcca, w \in L_1$ can look like this:

$$
\begin{aligned}
(s, agtcccttgtccca, S\#) \;& {}_e\Rightarrow (s, agtccttgtcca, SSSSS\#) && [5]\\
& {}_e\Rightarrow (s, agtccttgtcca, aSSSSS\#) && [2]\\
& {}_p\Rightarrow (s, gtccttgtcca, SSSSS\#) && \\
& {}_e\Rightarrow (s, gtccttgtcca, SStSSS\#) && [3]\\
& {}_e\Rightarrow (s, gtccttgtcca, SSttSSS\#) && [3]\\
& {}_e\Rightarrow (o, gtccttgtcca, SgSttSgSS\#) && [4]\\
& {}_e\Rightarrow (f, gtccttgtcca, SgtSttSgtSS\#) && [5]\\
& {}_e\Rightarrow (f, gtccttgtcca, SgtcSttSgtcSS\#) && [6]\\
& {}_e\Rightarrow (f, gtccttgtcca, SgtccSttSgtccSS\#) && [6]\\
& {}_e\Rightarrow (f, gtccttgtcca, SgtccSttSgtccSaS\#) && [7]\\
& {}_e\Rightarrow (f, gtccttgtcca, gtccttgtcca\#) && [7]\\
& {}_p\Rightarrow^* (f, \epsilon, \#) &&
\end{aligned}
$$

References

[1] Alexander Meduna.
Deep pushdown automata.
Acta Informatica, 2006(98):114–124, 2006.

radovan@klembara.pro