Arbirary CFPQ to Dyck language constrained querying

Semyon Grigorev
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034, Russia
semen.grigorev@jetbrains.com, rsdpisuy@gmail.com

This reduction is inspired by the construction described in [1].

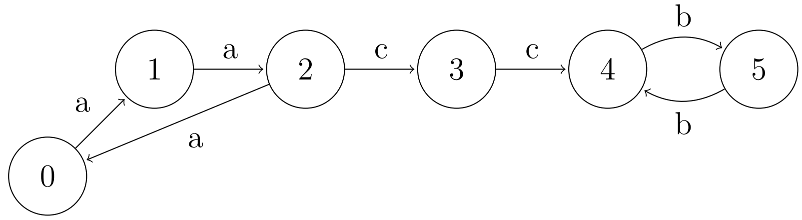
Consider a context-free grammar $\mathcal{G} = (\Sigma, N, P, S)$ in BNF where Σ is a terminal alphabet, N is a nonterminal alphabet, P is a set of productions, $S \in N$ is a start nonterminal. Also we denote a directed labeled graph by $G = (V, E, L)$ where $E \subseteq V \times L \times V$ and $L \subseteq \Sigma$.

We should construct new input graph G' and new grammar $\mathcal{G}'$ such that $\mathcal{G}'$ specifies a Dyck language and there is a simple mapping from CFPQ($\mathcal{G}', G'$) to CFPQ($\mathcal{G}, G$). Step-by-step example with description is provided below.

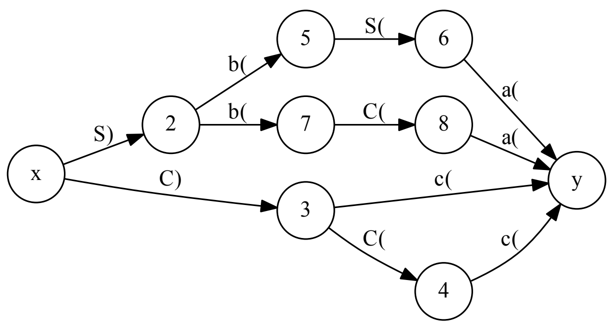
Let the input grammar is

$$
\begin{aligned}
S &\to a\ S\ b \mid a\ C\ b \\
C &\to c \mid C\ c
\end{aligned}
$$

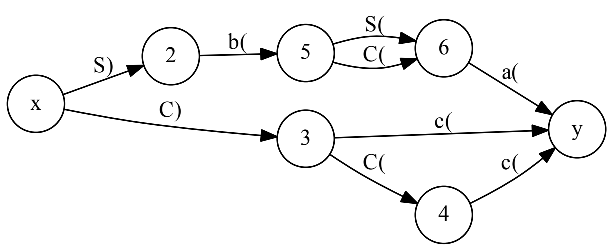
Let the input graph is



1. Let $\Sigma_{()} = \{t_(, t_) | t \in \Sigma\}$.
2. Let $N_{()} = \{N_(, N_) | N \in N\}$.
3. Let $M_{\mathcal{G}} = (V_{\mathcal{G}}, E_{\mathcal{G}}, L_{\mathcal{G}})$ is a directed labeled graph, where $L_{\mathcal{G}} \subseteq (\Sigma_{()} \cup N_{()})$. This graph is created the same manner as described in [1] but we do not require the grammar be in CNF. Let $x \in V_{\mathcal{G}}$ and $y \in V_{\mathcal{G}}$ is "start" and "final" vertices respectively. This graph may be treated as a finite automaton, so it can be minimized and we can compute an ε-closure if the input grammar contains ε productions. The graph $M_{\mathcal{G}}$ for our example is:

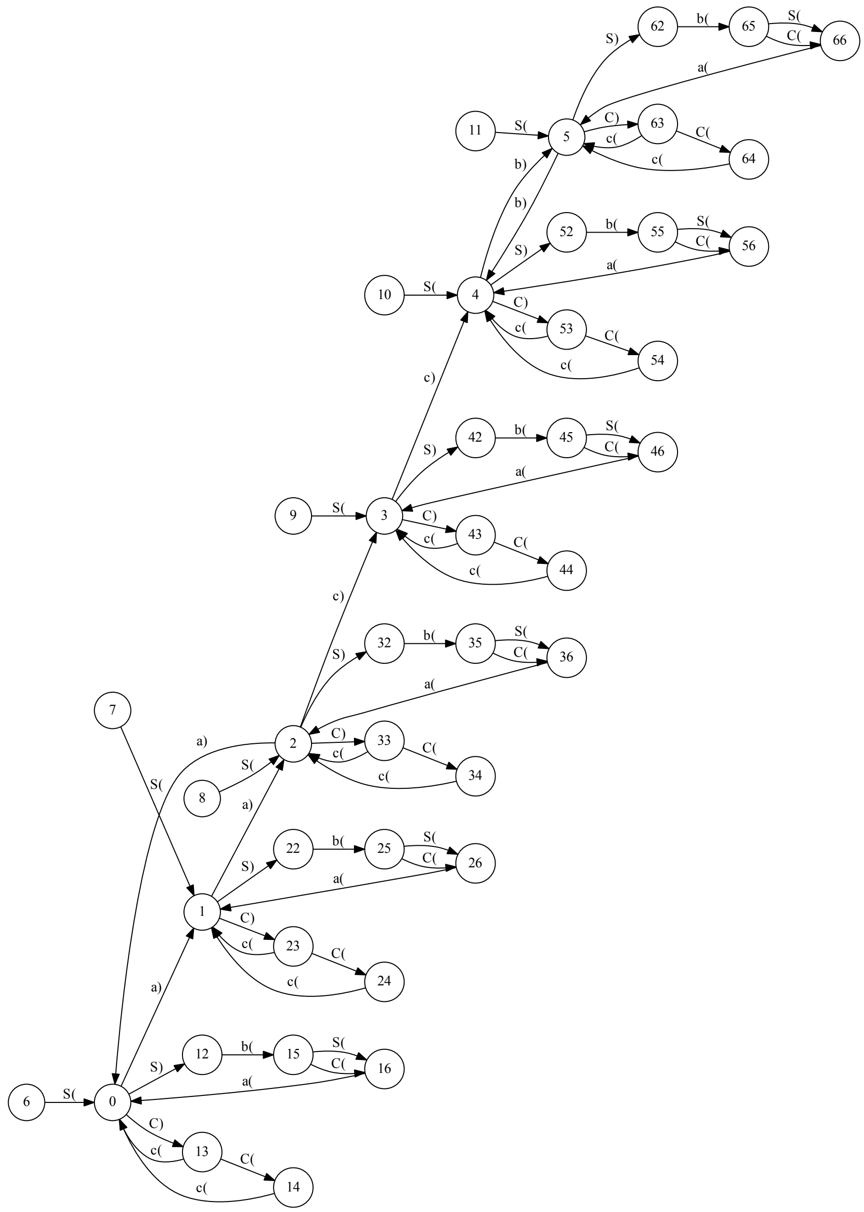
   The minimized graph:

4. For each $v \in V$ create $M_{\mathcal{G}}^v$: unique instance of $M_{\mathcal{G}}$.
5. New graph G' is a graph G where each label t is replaced with $t_)^i$ and some additional edges are created:

- Add an edge $(v', S_(, v)$ for each $v \in V$.
- And the respective $M_{\mathcal{G}}^v$ for each $v \in V$:
  - reattach all edges outgoing from x^v (“start” vertex of $M_{\mathcal{G}}^v$) to v;
  - reattach all edges incoming to y^v (“final” vertex of $M_{\mathcal{G}}^v$) to v.

  New input graph is ready:



6. New grammar $\mathcal{G}' = (\Sigma', N', P', S')$ where $\Sigma' = \Sigma_{()} \cup N_{()}$, $N' = \{S'\}$, $P' = \{S' \to b_(\; S' \; b_); S' \to b_(\; b_) \mid b_(, b_) \in \Sigma'\} \cup \{S' \to S' \; S'\}$ is a set of productions, $S' \in N'$ is a start nonterminal.

Now, if CFPQ($\mathcal{G}', G'$) contains a pair (u'_0, v') such that $e = (u'_0, S_{(}, u'_1) \in E'$ is an extension edge (step 5, first subitem), then $(u'_1, v') \in$ CFPQ($\mathcal{G}, G$). In our example, we can find the following path: $7 \xrightarrow{S_(} 1 \xrightarrow{S_)} 22 \xrightarrow{b_(} 25 \xrightarrow{C_(} 26 \xrightarrow{a_(} 1 \xrightarrow{a_)} 2 \xrightarrow{C_)} 33 \xrightarrow{C_(} 34 \xrightarrow{c_(} 2 \xrightarrow{c_)} 3 \xrightarrow{C_)} 43 \xrightarrow{c_(} 3 \xrightarrow{c_)} 4 \xrightarrow{b_)} 5$. Edge $7 \xrightarrow{S_(} 1$ is the extension, so (1,5) should be in CFPQ($\mathcal{G}, G$) and it is true.

1 Modified algorithm

Algorithm 1 Digraph flat exact paths

function DIGRAPH-FLAT-EXACT-PATHS(G)
  $\{{D_k}^{(-1)}, {D_k}^{(0)}, {D_k}^{(+1)}\} \leftarrow$ INIT-ADJACENCY-MATRICES(G)
  $M_1 \leftarrow$ AGMY-CODE-THEN-SUM(${D_1}^{(-1)}, {D_1}^{(0)}, {D_1}^{(+1)}$)
  ...
  $M_k \leftarrow$ AGMY-CODE-THEN-SUM(${D_k}^{(-1)}, {D_k}^{(0)}, {D_k}^{(+1)}$)
  $n \leftarrow |V|$
  for $l \in [2 \ldots \lceil \log n \rceil + 1]$ **do** ▷ Upper bound should be analized carefully
    $M' \leftarrow$ MARKUP-MINUS-ONE-EDGES(M_1) ▷ AGMY, mark -1 edges
    $M_1 \leftarrow M' \times M_1$ ▷ AGMY, non-Dyck 0 edges are detectable
    ... ▷ Do for all M_i
    $M_k \leftarrow M_k \times M_k$
    Remove ± 1 edges from all M_i
    $M_1 \leftarrow$ NORMALIZE-AND-DIVIDE-BY-2(M_1)
    ...
    $M_k \leftarrow$ NORMALIZE-AND-DIVIDE-BY-2(M_k)
    $Z \leftarrow$ GET-ZERO-EDGES(M_1)
    $Z \leftarrow Z +$ GET-ZERO-EDGES(M_2)
    ...
    $Z \leftarrow Z +$ GET-ZERO-EDGES(M_k)
    $M_1 \leftarrow Z \times M_1 \times Z$ ▷ AGMY, extend ± 1 and 0 edges for all M_i
    ...
    $M_k \leftarrow Z \times M_k \times Z$

References

[1] Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2017. *Optimal Dyck reachability for data-dependence and alias analysis.* Proc. ACM Program. Lang. 2, POPL, Article 30 (December 2017), 30 pages. DOI: https://doi.org/10.1145/3158118