D^3 as a 2-MCFL

Abstract. We discuss the open problem of parsing the Dyck language of 3 symbols, D^3, using a 2-Multiple Context-Free Grammar. We tackle this problem by implementing a number of novel techniques and present the associated software packages we developed.

Keywords: Dyck Language; multiple context free grammars (MCFG)

1 Introduction

Our goal with this paper is the analysis of the 3-dimensional Dyck Language, D^3, under the scope of a 2-multiple context-free language, 2-MCFL. For brevity's sake, this chapter only serves as a brief introductory guide towards relevant papers, where the interested reader will find definitions, properties and various correspondences of the problem.

D^3 is defined over a lexigraphically ordered alphabet (a, b, c) as the generalized language of well-balanced parentheses[**?**]. We attempt to model D^3 via a 2-MCFG; the class of multiple context free grammars that operate on pairs of strings[**?**].

There are a number of interesting correspondences to D^3. Firstly, a word of D^3 can be presented as a *standard Young Tableau*, which is a rectangular table with strictly ascending rows/columns containing as entries the numbers $\{1, 2, \ldots, n\}$ where n the total number of symbols in the word. The Young Tableau can be obtained by placing (in order) each character's index to the row corresponding to its lexicographical ordering (e.g. an a is placed on the first row)[**?**].

Another correspondence exists between D^3 and combinatorial *Spider Webs*, a special category of directed planar graphs embedded on a disk[**?**]. Spider Webs can be obtained through the application of a set of rules, known as the *Growth Algorithm*, which operates on pairs of neighbouring nodes, collapsing them into a singular intermediate node, transforming them into a new pair or eliminating them altogether. At termination, this process produces a well-formed Spider Web, which, in the context of D^3, can be interpreted as a visual representation of parsing a word.

A bijection links Young Tableaux with Spider Webs. Specifically, the *Jeu-de-taquin* algorithm can be applied on a Young Tableau, which transforms it to another one through an act called *promotion*. Subsequent promotions will eventually result in the initial tableau. Promotion thus defines an equivalence class, which we call an *orbit*[**?**].

2 Modeling Techniques

We now present a number of novel techniques that we developed as an attempt to solve the problem at hand, incrementally moving towards more complex and abstract grammars. For the purpose of experimentation we have implemented these techniques, based on a software library for parsing MCFGs[**?**]. The resulting Python code is open-source and available online[1].

2.1 Triple Insertion

To set things off, we start with the grammar of *triple insertion*. This grammar operates on non-terminals $\mathsf{W}(x, y)$, producing $\mathsf{W}(x', y')$ with an additional triplet *a, b, c* that respects the partial orders $x < y$ and $a < b < c$. The end-word is produced through the concatenation of (x, y).

$$\mathsf{S}(xy) \leftarrow \mathsf{W}(x, y). \quad (1)$$
$$\mathsf{W}(\epsilon, xy\mathbf{abc}) \leftarrow \mathsf{W}(x, y). \quad (2)$$
$$\ldots$$
$$\mathsf{W}(\mathbf{abc}xy, \epsilon) \leftarrow \mathsf{W}(x, y). \quad (61)$$
$$\mathsf{W}(\epsilon, \mathbf{abc}). \quad (62)$$
$$\ldots$$
$$\mathsf{W}(\mathbf{abc}, \epsilon). \quad (65)$$

Fig. 1. Grammar of triple insertions

$$\mathsf{S}(xy) \leftarrow \mathsf{W}(x, y).$$
$$\mathcal{O}_2[\![\mathsf{W} \leftarrow \epsilon \mid \{a < b < c\}]\!].$$
$$\mathcal{O}_2[\![\mathsf{W} \leftarrow \mathsf{W} \mid \{x < y,\ a < b < c\}]\!].$$

Fig. 2. $\mathcal{G}_0$: Meta-grammar of triple insertions

Despite being conceptually simple, this grammar consists of a large number of rules. Its expressivity is also limited; the prominent weak point is its inability to manage the effect of *straddling*, namely the generation of words whose substituents display complex interleaving patterns.

straddling graph

2.2 Meta-Grammars

To address the issue of rule size, we introduce the notion of *meta-grammars*, which allows a more abstract view of the grammar as a whole. Specifically, we define $\mathcal{O}$ as the *meta-rule* which, given a rule format, a set of partial orders (over the tuple indices of its premises and/or newly added terminal symbols), and the

[1] https://github.com/omelkonian/dyck

MCFG dimensionality, automatically generates all the order-respecting permutations. An example of how we can abstract away from explicitly enumerating the entirety of our initial rules is showcased in Fig. 2.

This approach enhances the potential expressivity of our grammars as well. For instance, we can now extend the previous with a single meta-rule that allows two non-terminals $\mathsf{W}(x, y)$, $\mathsf{W}(z, w)$ to interleave with one another, producing rearranged tuple concatenations and allowing some degree of straddling to be generated:

$$\mathcal{G}_1 : \mathcal{G}_0 + \mathcal{O}_2[\![\mathsf{W} \leftarrow \mathsf{W}, \mathsf{W} \mid \{x < y,\ z < w\}]\!].$$

The addition of this rule gets us closer to completeness, but we are still not there. We have thus far only used a single non-terminal, not utilizing the expressivity that an MCFG allows. To that end, we propose states to describe incomplete words; that is, words that either have an extra symbol or miss one. The former are positive states, whereas the latter are negative. The inclusion of these extra states would allow for more intricate interactions. Interestingly, there is a direct correspondence between these states and the nodes of Petersen's growth algorithm.

This meta-grammar, given below, consists of base cases for positive states, possible state interactions, closures of pairs of inverse polarity and a universally quantified meta-rule that allows the combination of any incomplete state with a well-formed one.

$$\begin{aligned}
&\mathsf{S}(xy) \leftarrow \mathsf{W}(x, y).\\
&\mathcal{O}_2[\![\mathsf{W} \leftarrow \epsilon \mid \{a < b < c\}]\!].\\
&\mathcal{O}_2[\![\mathsf{A}^+ \leftarrow \epsilon \mid \{a\}]\!].\\
&\mathcal{O}_2[\![\mathsf{B}^+ \leftarrow \epsilon \mid \{b\}]\!].\\
&\mathcal{O}_2[\![\mathsf{C}^+ \leftarrow \epsilon \mid \{c\}]\!].\\
&\mathcal{O}_2[\![C^- \leftarrow A^+, B^+ \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![B^- \leftarrow A^+, C^+ \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![A^- \leftarrow B^+, C^+ \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![A^+ \leftarrow C^-, B^- \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![B^+ \leftarrow C^-, A^- \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![C^+ \leftarrow B^-, A^- \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![W \leftarrow A^+, A^- \mid \{x < y < z < w\}]\!].\\
&\mathcal{O}_2[\![W \leftarrow C^-, C^+ \mid \{x < y < z < w\}]\!].\\
&\forall\, \mathsf{K} \in \{\mathsf{A}^{+/-},\ \mathsf{B}^{+/-},\ \mathsf{C}^{+/-}\} :\\
&\quad \mathcal{O}_2[\![\mathsf{K} \leftarrow \mathsf{K}, \mathsf{W} \mid \{x < y,\ z < w\}]\!].
\end{aligned}$$

Fig. 3. $\mathcal{G}_2$: Grammar of incomplete states

A further extension can be achieved through universally quantifying the notion of triple insertion, which is unique in the sense that it can insert three different terminals, each at a different position:

$$\mathcal{G}_3 : \mathcal{G}_2 + \forall\, \mathsf{K} \in \{\mathsf{A}^{+/-},\ \mathsf{B}^{+/-},\ \mathsf{C}^{+/-}\} : \mathcal{O}_2[\![\mathsf{K} \leftarrow \mathsf{K} \mid \{x < y,\ a < b < c\}]\!].$$

2.3 Rule Inference

The improved performance of the above approaches again proved insufficient to completely parse D^3. Our meta-rules are over-constrained by imposing a total order on the tuple elements, due to their inability to keep track of where the extra character(s) is. To overcome this, we split each state into multiple position-aware, *refined* states. Doing so revealed a vast amount of new interactions, as evidenced by the below alteration to the original A^+, B^+ interaction (where y can now occur after z or w):

$$\mathcal{O}_2[\![C^- \leftarrow A^+_{left}, B^+ \mid \{x < y, x < z < w\}]\!].$$

In order to accommodate the interactions between this increased number of states, we need to keep track of both internal and external order constraints. At this point, the abstraction offered by our meta-grammar approach does not cover our needs anymore. The same difficulty that we had encountered before is prominent once more, except now at an even higher level.

As a solution to the aforementioned limitation, we propose a system that can automatically create a full-blown m-MCFG given only the states it consists of. To accomplish this, we assign each state a unique *descriptor* that specifies the content of its tuple's elements. Aligning these descriptors with the tuple, we can then infer the descriptor of the resulting tuple of every possible state interaction. For the subset of those interactions whose resulting descriptor is matched with a state, we can now automatically infer the rule.

Formally, the system is initialized with a map $\mathcal{D}$, whose domain, $dom(\mathcal{D})$, is a set of *state identifiers* and its codomain, $codom(\mathcal{D})$, is the set of their corresponding *state descriptors* as illustrated in Fig.4.

$$\begin{aligned}
\mathsf{W} &\mapsto (\epsilon, \epsilon)\\
\mathsf{A}^+_l &\mapsto (a, \epsilon)\\
\mathsf{A}^+_r &\mapsto (\epsilon, a)\\
\mathsf{B}^+_l &\mapsto (b, \epsilon)\\
&\dots\\
\mathsf{B}^-_{l,r} &\mapsto (a, c)\\
\mathsf{C}^-_l &\mapsto (ab, \epsilon)\\
\mathsf{C}^-_r &\mapsto (\epsilon, ab)\\
\mathsf{C}^-_{l,r} &\mapsto (a, b)
\end{aligned}$$

Fig. 4. Map $\mathcal{D}$

Algorithm 1 ARIS: Automatic Rule Inference System

procedure aris($\mathcal{D}$)
  for $X \mapsto (d_1, \dots, d_n) \in \mathcal{D}$ **do**
    yield $X(d_1, \dots, d_n)$.
  for $X, Y \in dom(\mathcal{D})^2$ **do**
    $(X_{ord},\ Y_{ord}) \leftarrow (x < y < \dots,\ z < w < \dots)$
    for $(d_1, ..., d_n) \in \mathcal{O}_2[\![\_ \leftarrow X, Y \mid \{X_{ord}, Y_{ord}\}]\!]$ **do**
      for $S' \in$ eliminate$((d_1, \dots, d_n), \mathcal{D})$ **do**
        yield $S'(d_1, \dots, d_n) \leftarrow X, Y.$

procedure eliminate($(d_1, \dots, d_n), \mathcal{D}$)
  for $matches \in$ all_abc_triplets$(d_1, \dots, d_n)$ **do**
    for $i \in 0 \dots n/3$ **do**
      for $S' \in$ remove_abc_triplets$(matches, i)$ **do**
        if $S' \in codom(\mathcal{D})$ **then**
          yield S'

Meta-grammars accelarated the process of creating grammars, by letting us simply describe rules instead of explicitly defining them. ARIS builds upon this notion to raise the level of abstraction even further; one needs only specify a grammar's states and its descriptors, thus eliminating the need to define rules or even meta-rules.

2.4 Results

We display three charts, depicting the number of rules, percentage of counter-examples and computation times of each of our grammars for D_n^3 with n ranging from 2 to 6. The charts reveal two obvious trends; as grammars get more complex, the number of failing parses steadily declines. This however comes at the cost of rule size growth, which in turn is associated with dramatically exploding computation times. What this practically means is that we are unable to continue testing more elaborate grammars or scale our results to higher orders of n (recall that $\|D_n^3\|$ also has a very rapid rate of expansion).

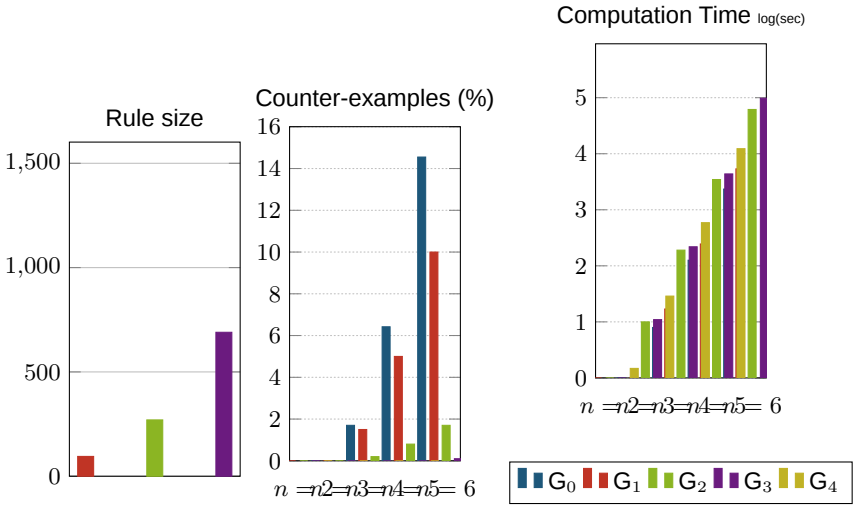

Fig. 5. Performance measures

3 Road to completeness

As the above results suggest, despite our best efforts the problem is yet unsolved. In this chapter we present a collection of unexplored ideas which we

consider noteworthy. These ideas are related to the concept of formally concluding the completeness of a grammar, bypassing the computational problem of parsing through words in search of an example the tested grammar fails to generate.

3.1 First-match policy and Relinking

Possibly the most intuitive way of checking whether a word w is part of D_n^3 is checking whether a pair of links occur that match a_i to b_i and b_i to c_i $\forall i \in n$, where x_i the i-th occurrence of symbol x in w. We call this process of matching the *first-match policy*.

Fig. 6. First-match policy for *"ababacbcabcc"*

The question arises whether a grammar can accomplish inserting a triplet of a, b, c in such a way that the triplet inserted always corresponds to a triplet matched by the first-match policy. If that were the case, it would be relatively easy to generalize this ability by induction to every $n \in \mathbb{N}$. Unfortunately, the answer is seemingly negative; the expressiveness provided by a 2-MCFG does not allow for the rather arbitrary insertions required. On a related note, being able to produce a word state $W(x, y)$ where $w = xy$ and x any possible prefix of w (i.e. all different comma positions i for $i \in 0, 1...3n$, gives no guarantee of being able to produce the same word with an extra triplet inserted due to the straddling property.

However, if rules existed that would allow for match-making and breaking, i.e. match *relinking*, an inserted symbol could be temporarily matched with what might be its first match-policy in a local scope, and then relink it to its correct match when merging two words together. This might be an interesting prospect to look into.

3.2 Growth Rules

The growth rules presented in section **??** are formally sound and complete. Part of them also have a very straightforward correspondence with some of our rules. For the other part, namely the growth rules which produce two edges, there is no direct way of translating them into a 2-MCFG. Adding memory to the grammar by adding states that describe two neighbouring edges might seem appealing but would not solve the problem; two neighbouring edges can in

turn interact with their next neighbours in a recursive fashion, which can not be accommodated by our current tuple representation. We cannot, however, rule out the possibility of the growth rules being scalably translatable into m-MCFG rules. In fact, such a translation would be a guarantee of completeness.

3.3 Insights from promotion

The act of promotion translates to cyclic rotation of the A_2 web [**?**]. Perhaps an interesting question here is whether promotion can be handled by a 2-MCFG (as a *context-free rewriting system*). If that is the case, it could be worth looking into the properties of orbits, to test for instance if there are promotions within an orbit that can be easier to solve than others. Solving a single promotion and transducing the solution to all other words belonging to the same orbit could then be a guideline towards completeness. The scale of such a task would of course be much greater, but the combination of these theoretical backbones into a unified body, combined with our mostly applied approach, could shed some light on the problem at hand.

4 Tools

4.1 Grammar utilities

We have implemented the modelling techniques we described in section 2 and distributed a Python package, called **dyck**, which essentially provides the programmer with a *domain-specific language* very close to this paper's mathematical notation. The following example demonstrates the similariry with G_1, defined in section **??**:

```
from dyck import *
G_1 = Grammar([
    ('S <- W', {(x, y)}),
    O('W', {(a, b, c)}),
    O('W <- W', {(x, y), (a, b, c)}),
    O('W <- W, W', {(x, y), (z, w)}),
])
```

It ships with several examples, showcasing meta-grammars and ARIS, as well as the complete set of implemented grammars defined in section **??**. The package provides a lot of features, relevant to experimentation on m-MCFGs, such as:

- Grammar selection
- Time measurements
- Rule/parse pretty-printing
- Dyck-specific utilities
  - Word generation
  - Soundness/completeness checking
- Grammar (de)serialization

4.2 Visualization

As counter-examples began to grow in size and number, we realised the necessity of a visualization tool to mitigate the challenging task of identifying certain properties they exhibit.

To that end, we distribute another Python package, called **dyckviz**, providing a command-line tool, which allows the simultaneous visualization of tableau-promotion and web-rotation (grouped in their corresponding equivalence classes).

Young tableaux in an orbit are colored via the first-match-policy, which sheds some light on how the *jeu-de-taquin* actually influences the structure of the correponding Dyck words. Interesting patterns have began to emerge, which still remain to be properly investigated.

Webs in a certain orbit, on the other hand, are rendered in a single PDF file (one web per page). Note that we represent them in a slightly simplified way compared to their definition in section 2, ignoring intermediate graph nodes that do not have corresponding non-terminals, as shown below:

5 Conclusion

We tried to accurately present the intricacies of D^3 and the difficulties that arise when attempting to model it under the scope of a 2-MCFL. We have developed and introduced some interesting techniques and tools, which we believe can be of use even outside the problem's narrow domain. We have largely expanded on the existing tool suite available through the MCFParser and made it easier for it to accommodate MIX-style languages and systems of meta-grammars in general.

Despite our best efforts, the question of whether D^3 can actually be encapsulated within a 2-MCFG still remains unanswered. Regardless, this problem has been very rewarding to pursue, and we hope to have intrigued the interested reader enough to further research the subject, use our code, or strive for a solution on his/her own.

Acknowledgements

We would like to thank Dr. Michael Moortgat for introducing us to the problem and getting us started with valuable bibliography and some initial code.

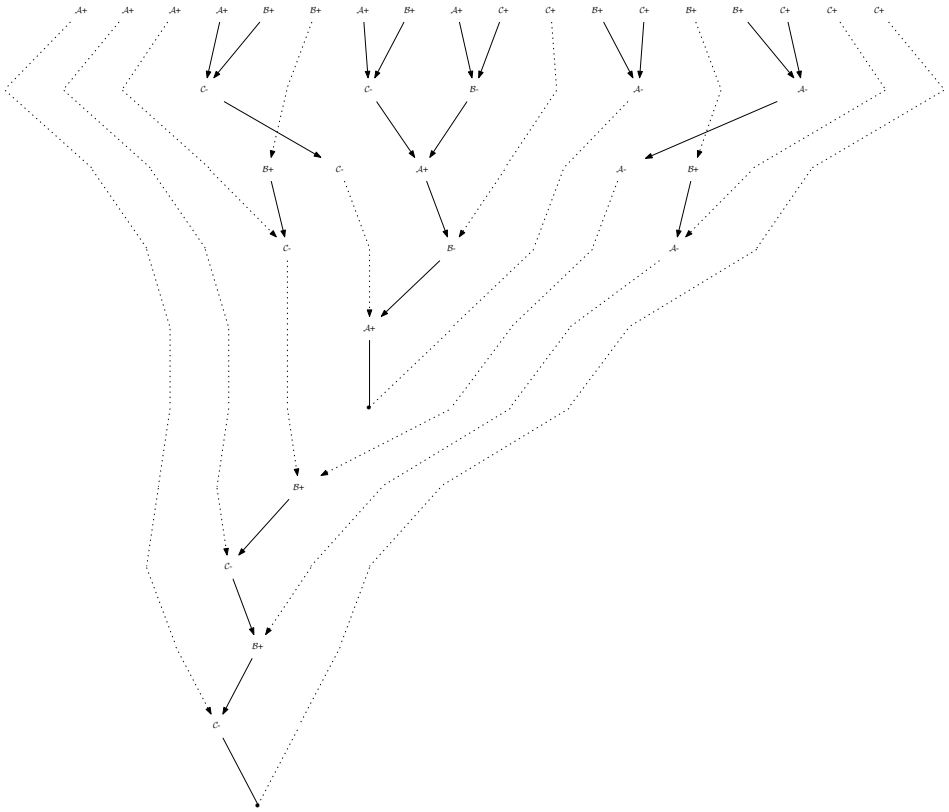

Fig. 7. Spider web of *"abaacbbacbabaccbcc"*