Analysis and Properties of Weighted DAG Automata

Eishitha Galpayage Don
ETH Zurich
egalpayage@student.ethz.ch

Abstract

In this paper, we study the weighted DAG automata of Chiang et al. [7], a recent formalism proposed to model semantic graphs. Blum and Drewes [6] establish many properties and algorithms of these automata for the unweighted case, such as closure properties, determinisation and minimisation. However, generalisations to the weighted case remains an open area of research, and this paper aims to start addressing these gaps. Indeed, algorithms and properties for weighted automata are important for future practical applications, such as the construction of probability models and inferences. In this paper, we will formalise and prove the recognition algorithm of [7], give algorithms for computing the weighted intersection and union, as well as provide a set of definitions and analysis methods in line with existing literature on weighted automata.

1 Introduction

String languages, grammars and automata have been extensively studied for decades, with standardised definitions, properties and algorithms. A core motivator of such research is the widespread applicability to language syntax and parsing, for both programming and natural languages. However, another important goal is the modeling of semantics. For instance, how do we represent the meaning of an English sentence? Abstract meaning representation [1] is a potential way of modelling meaning via semantic graphs. Formally, these semantic graphs are labelled, directed, acyclic graphs (DAGs). In order to describe and parse such structures, we require formalisms for DAG languages, of which many have been proposed [10; 12; 11], each with various idiosyncrasies.

The main goal of this project is to investigate theoretical properties and algorithms admitted by weighted Directed Acyclic Graph (DAG) languages and automata. The existing literature for weighted automata over strings and trees share many common ideas and concepts. For instance, forward and backward weights have been defined for both string and tree automata, representing essentially the sum of paths leading to or emanating from a given state. This allows us to compute marginals for transitions, and convert the weight distribution of transitions emanating from a state into a probability distribution. Such concepts have applications in machine learning and probabilistic inference. [7] outlines a related notion of 'partial analysis' for nodes in a DAG, and we aim to formalise this in a similar fashion to forward and backward weights.

There exists ample evidence in the literature to demonstrate that this topic is still present in contemporary research and there is some potential for extending the existing frameworks. For example, Blum and Drewes explores language-theoretic properties of DAG languages and provide an explicit analysis of Chiang et al.'s formalism. In general however, the papers focused on the unweighted case. One of our aims is to begin the process of generalising these results and definitions to the weighted case.

2 Literature review

2.1 Semantic Graphs

The Penn Treebank is a corpus of English natural language sentences paired with syntactic trees, and is widely used in syntactic modelling. However, the semantics of a sentence often involves more complex relations that cannot be modelled as simple rooted trees. Abstract Meaning Representation [1] is an attempt to formalise a system for semantic representation, in which thousands of English sentences are

broken down into their semantics and expressed as a semantic graph. Semantic graphs as used in AMR are specified as rooted, directed, acyclic graphs with both node (leaf) and edge labels [1]. It also aims to be independent of syntax, e.g "he described her as a genius" has an identical semantic graph to "she was a genius according to his description".

2.2 DAG Automata and Grammars

The concepts of DAG automata and grammars have existed for a while, the most cited being those of Kamimura and Slutzki [10], Quernheim and Knight [12], and Priese [11]. These have all been born out of different needs and use cases, and as such have quite different properties. For instance, the formalism of [10] recognises only planar DAGs, and is hence unsuitable for modelling semantic graphs.

More recently, there have been two main formalisms proposed specifically with semantic modelling in mind. These are the weighted DAG automata of [7], which are studied in this paper, and the order preserving DAG grammars of [9]. The formalisms differ in that the DAG automata recognise only vertex labelled graphs, while OPDGs generate edge labelled hypergraphs, but both formalisms provide simple transformations converting to and from semantic graphs.

2.2.1 Weighted DAG Automata

The seminal paper by [7] describes the DAG automaton as a simplified version of the formalism proposed by [12]. Most notably, the authors provide proofs of a number of properties. For instance, they posit that path languages of DAG automata are regular under the constraint that there exist multiple roots. Furthermore, they demonstrate that hyperedge replacement graph languages are closed under intersection with languages recognised by DAG automata. They also show that testing for emptiness of DAG automata is decidable under the definition assumed in the present paper, but not under the original definition by [10], and that general recognition is NP-complete, even in the non-uniform case (via a reduction from SAT). It is also interesting to mention that the authors define 'extended weighted DAG automata', which allow rules that can process nodes with unbounded degree. Essentially, these rules/transitions use a form of regular expressions to match structures.

[6] then analyses the language theoretic properties of these unweighted DAG automata. Namely, they show that recognisable/regular DAG languages (DAG languages recognised by DAG automata) are closed under union and intersection, but not under complement. It is also shown that deterministic DAG automata are strictly less powerful than their nondeterministic counterparts. A pumping lemma is then presented for regular DAG languages, as well as a characterisation of infinite DAG languages in terms of cyclic automata.

Finally, for deterministic DAG automata, [6] presents a definition of equivalent and distinguishable states, a minimisation procedure, and a polynomial time method for computing the equivalence of deterministic DAG automata.

2.2.2 Order Preserving DAG Grammars

Weighted order-preserving DAG grammars (WOPDGs) on the other hand are a generative model.

The formalism for OPDGs introduced by [9] is based on introducing several constraints on the original definition of weighted DAG grammars. OPDGs have been subsequently studied in a series of papers. For instance, [4] gives a polynomial time algorithm for parsing OPDGs and mention NP-hard result for grammars obeying a subset of the original restrictions. [2] establishes an algebraic characterisation of OPDLs, defining operations such as concatenation, where the languages induced by the algebra are exactly the languages generated by OPDGs. This allows for Nerode congruence and the Myhill-Nerode theorem.

Further properties are explored in [3], who define weighted OPDLs over commutative semirings. Further, a link between WOPDGs and weighted tree grammars is established. This allows the minimisation problem for deterministic WOPDGs to be reduced to the case for a weighted tree grammar. They then give asymptotic bounds for the complexity of minimisation. OPDLs are also proved to be MSO-definable (that is, given a grammar, we can construct a monadic second-order formula such that if g is generated by the grammar, the formula is a logical consequence of g). However, they do not show the reverse: given an

[1] https://www.isi.edu/~ulf/amr/help/amr-guidelines.pdf

MSO formula that defines a graph language (that is within our 'universe'), we can construct an OPDG that represents the same language.

As for parsing, [4] proposes a uniform polynomial parsing algorithm for a generalisation of OPDGs. Uniform is used here in the sense that the grammar is considered as a part of the input, while non-uniformity assumes a fixed grammar. Finally, [5] extends the uniform polynomial parsing algorithm to the weighted case.

3 Automata Definitions and Preliminaries

We now present the definition of DAG automata as per [7] and [6], starting with some preliminaries.

An *alphabet* Σ is a finite, nonempty set of symbols. A *ranked alphabet* is a tuple $(\Sigma, rank)$, where Σ is an alphabet, and $rank : \Sigma \to \mathbb{N}$ is a ranking function. A *doubly ranked* alphabet has a ranking function $rank : \Sigma \to \mathbb{N} \times \mathbb{N}$, where each symbol is assigned a *head rank* and a *tail rank*.

Given a set S, a *multiset* over S is a mapping $\mu : S \to \mathbb{N}$. Informally, it is a set where elements can appear multiple times, and we will simply write it as $\{x_1, x_2, \ldots, x_k\}$ where the x_i need not be distinct. We denote the collection of all multisets over set S as $\mathcal{M}(S)$.

A *commutative semiring* is a tuple $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ such that $(\mathbb{K}, \oplus)$ is a commutative monoid with identity $\mathbf{0}$, $(\mathbb{K}, \otimes)$ is a commutative monoid with identity $\mathbf{1}$, $\otimes$ left and right distributes over $\oplus$, and $\otimes$ with $\mathbf{0}$ annihilates $\mathbb{K}$. In this paper, when we refer to semirings, we mean commutative semirings and we will simply denote the structure by $\mathbb{K}$. Further, a semiring is *zero sum free* if there are no elements $a, b \in \mathbb{K} \setminus \{\mathbf{0}\}$ such that $a \oplus b = \mathbf{0}$, and a semiring is *zero divisor free* if there are no elements $a, b \in \mathbb{K} \setminus \{\mathbf{0}\}$ such that $a \otimes b = \mathbf{0}$.

3.1 Σ-Graphs and DAGs

Definition 3.1 (Σ-Graph)**.** *Let Σ be a doubly ranked alphabet of node labels. A Σ-graph over is a tuple $G = (V, E, lab_V, in, out)$ such that:*

- *V and E are finite sets of vertices and edges respectively.*
- *$lab_V : V \to \Sigma$ is a node labelling function.*
- *$in : V \to E^*$ is a function mapping each vertex to an ordered sequence of incoming edges (possible empty).*
- *$out : V \to E^*$ is a function mapping each vertex to an ordered sequence of outgoing edges (possible empty).*

We additionally require that $(|in(v)|, |out(v)|) = rank(lab_V(v))$, i.e the node's incoming/outgoing edges match the rank of the label. Moreover, we require that every edge $e \in E$ has $e \in in(v)$ for exactly one v, and $e \in out(v')$ for exactly one v', and occurs in these two sequences exactly once. We refer to v, v' as $tar(e), src(e)$ respectively.

From this point on we shall refer to Σ-graphs as just graphs.

We say that a node v is a *root* node if $in(v) = \lambda$, where λ represents the empty sequence. Likewise, a *leaf* node v has $out(v) = \lambda$.

Definition 3.2 (Graph Fragment)**.** *We define a r-graph fragment (with $r = (h, t)$) as a tuple $\boldsymbol{d}_r = (V, E, lab_V, in, out, E_h, E_t)$ where (V, E, lab_V, in, out) is a Σ-graph. $E_h = e_1 e_2 \ldots e_h$ is an ordered set of 'head' edges with no source, but a target in V, and this set is disjoint from E. $E_t = e'_1 e'_2 \ldots e'_t$ is likewise an ordered set of 'tail' edges with no target, again disjoint from both E and E_t. A $(0, 0)$-graph fragment is just a Σ-graph.*

Intuitively, an r-graph fragment is a graph with h 'dangling' incoming edges and t 'dangling' outgoing edges for $r = (h, t)$.

A *path* in a graph or graph fragment is a sequence of edges $e_1 e_2 \ldots e_k$ such that $tar(e_i) = src(e_{i+1})$ for $1 \leq i \leq k - 1$. A *cycle* is a path such that $tar(e_k) = src(e_1)$.

A node u is a *descendant* of node v if there exists a nonempty path $e_1 e_2 \dots e_k$ such that $src(e_1) = v$ and $tar(e_k) = u$. We also say that v is an *ancestor* of u. Nodes are descendants/ancestors of themselves.

Definition 3.3 (DAG and DAG Languages)**.** *A Σ-DAG (henceforth just referred to as a DAG) is a Σ-graph with no cycles (and a DAG fragment is a graph fragment with no cycles). Denote by $\mathcal{D}_\Sigma$ the set of all nonempty connected DAGs over Σ. A DAG language L is a subset of $\mathcal{D}_\Sigma$. Also denote by $\mathcal{H}_\Sigma$ the set of nonempty connected DAG fragments over Σ.*

3.2 Automata Definition

In this section we focus on the DAG automata studied by [7] and [6], but primarily use the definitions and notation of the latter. The main difference is that the latter enforces an ordering of the incoming and outgoing edges on each node, and thus transitions are defined on sequences of states as opposed to multisets. Further, Chiang's definition uses an unranked alphabet. These differences are minor, as given an automaton as per Chiang's definition, we can represent it by adding multiple transitions corresponding to each possible ordering of the state multisets. The unranked alphabet can be converted to a doubly ranked alphabet by taking all the transitions for each symbol a and creating ranked symbol(s) $a_{(h,t)}$ for each transition taking h states to t states.

Definition 3.4 (DAG Automaton)**.** *A weighted DAG automaton is a tuple $M = \langle \Sigma, Q, \Theta, \delta, \mathbb{K} \rangle$ where:*

- *Σ is a doubly ranked alphabet of node labels*
- *Q is a finite set of states*
- *$\mathbb{K}$ is a commutative semiring*
- *$\Theta \subseteq Q^* \times \Sigma \times Q^*$ is a set of transitions, where each transition $(q_1 q_2 \dots q_m, \sigma, p_1 p_2 \dots p_n)$ is written $\alpha \xrightarrow{\sigma} \beta$, where $m, n \geq 0$.*
- *$\delta : Q^* \times \Sigma \times Q^* \to \mathbb{K}$ is a weight function assigning a weight to each transition. If $\delta(t) = w$, we also write the transition t as $\alpha \xrightarrow{\sigma/w} \beta$. Note that for transitions not in Θ, we define the weight as semiring 0.*

Transitions take a sequence of states and an alphabet symbol and output another sequence of states. This corresponds to reading a DAG node (labelled with σ) and associating states with the incoming and outgoing edge sequences (using the implied sequence ordering). Further, we write transitions on root nodes as $\lambda \xrightarrow{\sigma/w} \beta$, and transitions on leaf nodes as $\alpha \xrightarrow{\sigma/w} \lambda$.

We now define a **run** of automaton M on DAG D as a mapping from edges to states, $\rho : E \to Q$. This lets us extend the weighting function to runs, where we define the **run weight** of ρ to be[2]:

$$\delta(\rho) = \bigotimes_{v \in V} \delta(\rho(in(v)), lab_V(v), \rho(out(v)))$$

We also define the **graph weight** of D under M, denoted by $[[M]](D)$ in [7]:

$$[[M]](D) = \bigoplus_{\text{all runs } \rho \text{ on } D} \delta(\rho)$$

Observe that unweighted DAG automata correspond to weighted DAG automata over the Boolean semiring, and the DAG language **recognised** by unweighted/Boolean semiring weighted DAG automaton M is the set $L(M) = \{D \in \mathcal{D}_\Sigma \mid [[M]](D) = true\}$. The set of **recognisable DAG languages** consists of languages for which there exists an unweighted DAG automaton recognising it.

As pointed out in [6], $L(M)$ only contains nonempty, connected DAGs, since we define it over $\mathcal{D}_\Sigma$. However, observe that for $D_1, D_2 \in L(M)$, then the disconnected DAG written as $D = D_1 \& D_2 =$

[2]When we write $\rho(in(v))$ or $\rho(out(v))$, we mean the sequence $\rho(e_1)\rho(e_2)\dots\rho(e_k)$ where $in(v) = e_1 e_2 \dots e_k$ is the sequence of incoming edges

$(V_1 \cup V_2, E_1 \cup E_2, lab_{V_1} \cup lab_{V_2}, in_1 \cup in_2, out_1 \cup out_2)$ is also accepted by M assuming that the vertex and edge sets are disjoint. Indeed, we can construct an accepting run $\rho_D = \rho_1 \cup \rho_2$ given accepting runs ρ_1, ρ_2 for D_1 and D_2 respectively. Hence, if M accepts a language of nonempty, connected DAGs $L(M)$, it also accepts the infinite language $L(A)^{\&}$, defined as the set of all DAGs of the form $D_1 \& D_2 \& \ldots \& D_n$ for $D_i \in L(A)$ and $n \geq 0$ (where $n =$ corresponds to the empty DAG). We can think of this as similar to a Kleene closure over the set of connected DAGs $L(M)$, although in this case $L(M)$ may be infinite.

Observation 3.5. *Given disjoint $D_1, D_2 \in L(M)$, we have $D = D_1 \& D_2$ such that $[[M]](D) = [[M]](D_1) \otimes [[M]](D_2)$*

Proof. Each run of M on D can be decomposed into the union of a run on D_1 and a run on D_2, as the edge sets are disjoint. As such, if ρ is a run on D, it follows from the definition of $\delta(\rho)$ and the disjointness of the vertex sets that $\delta(\rho) = \delta(\rho_1) \otimes \delta(\rho_2)$ where ρ_1, ρ_2 are runs on D_1, D_2 respectively. This gives us:

$$\begin{aligned}
[[M]](D) &= \bigoplus_{\text{all } \{\rho_1, \rho_2\}} \delta(\rho_1) \otimes \delta(\rho_2) \\
&= \bigoplus_{\text{all } \rho_1} \bigoplus_{\text{all } \rho_2} \delta(\rho_1) \otimes \delta(\rho_2) \\
&= \bigoplus_{\text{all } \rho_1} \left(\delta(\rho_1) \otimes \bigoplus_{\text{all } \rho_2} \delta(\rho_2) \right) \\
&= \left(\bigoplus_{\text{all } \rho_1} \delta(\rho_1) \right) \otimes \left(\bigoplus_{\text{all } \rho_2} \delta(\rho_2) \right) \\
&= [[M]](D_1) \otimes [[M]](D_2)
\end{aligned}$$

□

4 Contexts, Partial Weights, and a Formal Correctness Proof of the Recognition Algorithm

In this section, we define the notion of graph contexts and partial weights (a similar concept to forward/backward weights of string automata). This was inspired by the recognition/graph weight calculation algorithm presented by [7] and we use these definitions to formalise and prove correctness of the algorithm.

Definition 4.1 (Graph Context)**.** *A graph context is a function $\mathbf{c} : \mathcal{H}_\Sigma^k \to \mathcal{H}_\Sigma$, that takes in k graph fragments and returns another graph fragment. Every graph context $\mathbf{c}$ has a type $type(\mathbf{c}) = \langle (r_1, r_2, \ldots, r_k), r' \rangle$ such that the i'th argument fragment is of type r_i, and the returned graph fragment is of type r'. Further, a context has sequences of incoming and outgoing edges $E_h = e_1 e_2 \ldots e_h, E_t = e'_1 e'_2 \ldots e'_t$ for $r' = (h, t)$.*

We define the set of graph contexts C_Σ over doubly ranked alphabet Σ as the smallest set defined inductively according to the following:

- *Consider the function $\mathbf{c}$ of type $\langle (r), r \rangle$ such that $\mathbf{c}[\mathbf{d}_r] = \mathbf{d}_r$ for any r-graph fragment. Then $\mathbf{c} \in C_\Sigma$, and E_h, E_t are the same as those of $\mathbf{d}_r$. These are 'trivial' contexts and can be thought of as a placeholder/gap.*
- *Let $\mathbf{d}_r$ be a r-graph fragment consisting of k nodes, and let $v_1, v_2, \ldots, v_k$ be a sequence of these nodes whose labels have ranks $r'_1, r'_2, \ldots, r'_k$ respectively. Let $\mathbf{c}_i \in C_\Sigma$ be a context of type $\langle (r''_{1_i}, \ldots, r''_{m_i}), r'_i \rangle$. Then, let $\mathbf{c}$ be the function obtained by replacing the nodes $v_1, v_2, \ldots v_k$ in $\mathbf{d}_r$ with contexts $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$ (matching the edges based on the ordering). Then, $\mathbf{c} \in C_\Sigma$ and $\mathbf{c}$ is a context with rank $\langle (r''_{1_1}, \ldots, r''_{m_1}, \ldots, r''_{1_k}, \ldots, r''_{m_k}), r \rangle$. The incoming and outgoing edge sequences E_h, E_t are the same as those of $\mathbf{d}_r$, and some of these edges may also be identified with edges of an internal context $\mathbf{c}_i$. Informally speaking, taking a graph fragment and replacing all vertices with contexts generates a new context.*

Let $\boldsymbol{c}$ be a context of type $\langle (r_1, r_2, \ldots, r_k), r' \rangle$. Given graph fragments $\boldsymbol{d}_{r_1}, \boldsymbol{d}_{r_2}, \ldots, \boldsymbol{d}_{r_k}$ we refer to the term $\boldsymbol{c}[\boldsymbol{d}_{r_1}, \boldsymbol{d}_{r_2}, \ldots, \boldsymbol{d}_{r_k}]$ as a grounded context*, and such a term is a r'-graph fragment. When performing this substitution, each 'argument' of the context $\boldsymbol{c}$ is replaced with a graph fragment of matching rank, and the incoming/outgoing edges are identified with each other based on the ordering.*

This definition also implies that contexts can be nested (provided the types agree). For instance, a context $\boldsymbol{c}$ with type $\langle (r, r'), r \rangle$ can be nested into its first argument to form $\boldsymbol{c}' = \boldsymbol{c}[\boldsymbol{c}, \boldsymbol{c}_{r'}]$, which is a $\langle (r, r', r'), r \rangle$ context (where we use the trivial context $\boldsymbol{c}_{r'}$ of type $\langle (r'), r' \rangle$ for completeness).

Furthermore, observe that this construction is similar to a grammar-like derivation of DAGs, starting with a trivial context of type $\langle (r), r \rangle$ where $r = (0, 0)$. Graphs could then be generated with rules that replace contexts with fragments/other contexts, with fragments representing terminals and contexts representing nonterminals. This is similar to node replacement graph grammars and relationships between such grammars and automata may be worth looking into.

Lemma 4.2. *Every graph fragment can be expressed as at least one grounded context.*

Proof. Let $\boldsymbol{d}_r$ be an r-graph fragment with n nodes. Given some ordering of the nodes $1, \ldots, n$, if we replace every node i of rank r_i with a trivial context of type $\langle (r_i), r_i \rangle$, we obtain a context $\boldsymbol{c}$ with arguments corresponding to each node of $\boldsymbol{d}_r$ and an output type of r, as per the inductive definition of a graph context. Let $\boldsymbol{d}^{(i)}_{r_i}$ be the r_i-graph fragment containing just node i with label rank r_i. Then, since the arguments of $\boldsymbol{c}$ are in the node order of $1, \ldots, n$, it is easy to see that $\boldsymbol{c}[\boldsymbol{d}^{(1)}_{r_1}, \boldsymbol{d}^{(2)}_{r_2}, \ldots, \boldsymbol{d}^{(n)}_{r_n}] = \boldsymbol{d}_r$. □

For graph fragments with more than 1 node, there are multiple ways to express the fragment as a grounded context.

Definition 4.3 (Partial Weight)**.** *Let $M = \langle \Sigma, Q, \Theta, \delta, \mathbb{K} \rangle$ be a DAG automaton, and let $\boldsymbol{d}_r$ be a graph fragment with $r = (h, t)$. We define the* ***partial weight*** $\boldsymbol{\alpha}$ *of the graph fragment given a mapping $f : E_h \cup E_t \to Q$ inductively as follows:*

- *For a graph fragment of 1 node v, we have*

$$\boldsymbol{\alpha}[\boldsymbol{d}_r, f] = \delta(f(E_h), lab_V(v), f(E_t))$$

- *For a graph fragment with $n > 1$ nodes, by the construction in Lemma 4.2, we can express $\boldsymbol{d}_r = \boldsymbol{c}[\boldsymbol{d}^{(1)}_{r_1}, \boldsymbol{d}^{(2)}_{r_2}, \ldots, \boldsymbol{d}^{(n)}_{r_n}]$. We then define:*

$$\boldsymbol{\alpha}[\boldsymbol{d}_r, f] = \boldsymbol{\alpha}[\boldsymbol{c}[\boldsymbol{d}^{(1)}_{r_1}, \boldsymbol{d}^{(2)}_{r_2}, \ldots, \boldsymbol{d}^{(n)}_{r_n}], f] = \bigoplus_{all\ \mathcal{F}(f)} \bigotimes_{i=1}^{n} \boldsymbol{\alpha}[\boldsymbol{d}^{(i)}_{r_i}, f_i]$$

  Where $\mathcal{F}(f)$ is an (ordered) set of mappings $\{f_1, f_2, \ldots, f_n\}$ such that[3] $f_i : E_{r_i[0]} \cup E_{r_i[1]} \to Q$, which maps the 'dangling' edges of $\boldsymbol{d}^{(i)}_{r_i}$ to states. We require that the set of these mappings has no conflicts, both amongst the f_i's and with f, such that if we were to draw the full graph fragment $\boldsymbol{d}_r$ and assign states to edges based on the functions in $\mathcal{F}(f)$, each edge would have exactly one state. More formally, we must obtain a function $\rho : E \cup E_h \cup E_t \to Q$ for $\boldsymbol{d}_r = (V, E, lab_V, in, out, E_h, E_t)$ such that $\rho = \bigcup_i \mathcal{F}(f)_i$ and ρ agrees with f on $E_h \cup E_t$.

- *We also extend the above definition for the case where $\boldsymbol{d}_r$ is expressed as a different grounded context, i.e $\boldsymbol{d}_r = \boldsymbol{c}[\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k]$, where $\boldsymbol{t}_i$ is some grounded context:*

$$\boldsymbol{\alpha}[\boldsymbol{c}[\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k], f] = \bigoplus_{all\ \mathcal{F}(f)} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}[\boldsymbol{t}_i, \mathcal{F}(f)_i]$$

[3] If $r = (x, y)$ we say $r[0] = x, r[1] = y$

We now show that the above definition is consistent, i.e two identical graph fragments expressed as different grounded contexts will have the same partial weight, and that it aligns with the definition of graph weight.

Lemma 4.4. *Let $\boldsymbol{d}_r$ be an r-graph fragment with $n > 1$ nodes. We have from Lemma 4.2 that $\boldsymbol{d}_r = \boldsymbol{c}_1[\boldsymbol{d}_{r_1}^{(1)}, \boldsymbol{d}_{r_2}^{(2)}, \ldots, \boldsymbol{d}_{r_n}^{(n)}]$. Let $\boldsymbol{d}_r = \boldsymbol{c}_2[\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k]$ be another grounded context equivalent to $\boldsymbol{d}_r$, where $\boldsymbol{t}_i$ is a grounded context. We have that:*

$$\boldsymbol{\alpha}[\boldsymbol{c}_1[\boldsymbol{d}_{r_1}^{(1)}, \boldsymbol{d}_{r_2}^{(2)}, \ldots, \boldsymbol{d}_{r_n}^{(n)}], f] = \boldsymbol{\alpha}[\boldsymbol{c}_2[\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k], f]$$

Proof. Each $\boldsymbol{t}_i$, being a grounded context, is itself a graph fragment and can therefore be expressed as $\boldsymbol{c}_i[\boldsymbol{d}_{r_{i,1}}^{(i,1)}, \boldsymbol{d}_{r_{i,2}}^{(i,2)}, \ldots, \boldsymbol{d}_{r_{i,m_i}}^{(i,m_i)}]$ as in Lemma 4.2 such that $\boldsymbol{d}_{r_{i,1}}^{(i,1)}$ is a single node. Then, since the $\boldsymbol{t}_i$'s form a partition of $\boldsymbol{d}_r$'s nodes, the union of all the nodes of the $\boldsymbol{t}_i$'s is exactly the set of nodes in $\boldsymbol{d}_r$, i.e $\bigcup_{i=1}^{k}\bigcup_{j=1}^{m_i}\{\boldsymbol{d}_{r_{i,j}}^{(i,j)}\} = \bigcup_{i=1}^{n}\{\boldsymbol{d}_{r_i}^{(i)}\}$.

We have that:

$$\begin{aligned}
\boldsymbol{\alpha}[\boldsymbol{c}_2[\boldsymbol{t}_1, \ldots, \boldsymbol{t}_k], f] &= \bigoplus_{\text{all } \mathcal{F}(f)} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}[\boldsymbol{t}_i, f_i] \\
&= \bigoplus_{\text{all } \mathcal{F}(f)} \bigotimes_{i=1}^{k} \left(\bigoplus_{\text{all } \mathcal{F}'(f_i)} \bigotimes_{j=1}^{m_i} \boldsymbol{\alpha}[\boldsymbol{d}_{r_{i,j}}^{(i,j)}, f'_{i,j}] \right)
\end{aligned}$$

Note that f_i is a mapping of dangling edges of $\boldsymbol{t}_i$ to states in Q, and $\mathcal{F}'(f_i)$ is another set of non-conflicting mappings $\{f'_{i,1}, \ldots, f'_{i,m_i}\}$ of which $f'_{i,j}$ maps dangling edges of node $\boldsymbol{d}_{r_{i,j}}^{(i,j)}$ to states in Q.

We can then express the above equation as:

$$\begin{aligned}
\boldsymbol{\alpha}[\boldsymbol{c}_2[\boldsymbol{t}_1, \ldots, \boldsymbol{t}_k], f] &= \bigoplus_{\text{all } \mathcal{F}(f)} \bigotimes_{i=1}^{k} \left(\bigoplus_{\text{all } \mathcal{F}'(f_i)} \bigotimes_{j=1}^{m_i} \boldsymbol{\alpha}[\boldsymbol{d}_{r_{j_i}}^{(j_i)}, f'_{i,j}] \right) \\
&= \bigoplus_{\text{all } \mathcal{F}(f)} \bigoplus_{\text{all } \mathcal{F}'(f_1), \ldots, \mathcal{F}'(f_k)} \bigotimes_{i=1}^{k} \bigotimes_{j=1}^{m_i} \boldsymbol{\alpha}[\boldsymbol{d}_{r_{j_i}}^{(j_i)}, f'_{i,j}] \\
&= \bigoplus_{\text{all } \mathcal{F}^*(f)} \bigotimes_{i=1}^{n} \boldsymbol{\alpha}[\boldsymbol{d}_{r_i}^{(i)}, f_i^*] \\
&= \boldsymbol{\alpha}[\boldsymbol{c}_1[\boldsymbol{d}_{r_1}^{(1)}, \boldsymbol{d}_{r_2}^{(2)}, \ldots, \boldsymbol{d}_{r_n}^{(n)}], f]
\end{aligned}$$

Where given some $\mathcal{F}(f)$, we define $\mathcal{F}^*(f) = \bigcup_{i=1}^{k} \mathcal{F}'(f_i) = \bigcup_{i=1}^{k}\bigcup_{j=1}^{m_i}\{f'_{i,j}\} = \{f_1^*, f_2^*, \ldots, f_n^*\}$ as from above we know that the $\boldsymbol{t}_i$'s partition the nodes of $\boldsymbol{d}_r$ (and assuming some consistent ordering). □

Lemma 4.5. *Let M be a DAG automaton. For a graph fragment $\boldsymbol{d}_r$ with $r = (0, 0)$, i.e a standard DAG, we have:*

$$\boldsymbol{\alpha}[\boldsymbol{d}_r, \emptyset] = [[M]](\boldsymbol{d}_r)$$

Note: since a $(0, 0)$ ranked graph fragment has no dangling edges, we define the only possible mapping from its dangling edges to states as $\emptyset$. This also means that $\mathcal{F}(\emptyset)$ need only be internally consistent amongst the f_i's.

Proof. Let n be the number of nodes in $\boldsymbol{d}_r$. Since $\mathcal{F}(\emptyset) = \{f_1, f_2, \ldots f_n\}$ corresponds to a set of non-conflicting mappings from edges to states, taking the union of all the mappings gives us a run

$\rho : E \to Q$, where E is the set of edges of $\boldsymbol{d}_r$. Again using the construction from Lemma 4.3 to get $\boldsymbol{d}_r = \boldsymbol{c}[\boldsymbol{d}_{r_1}^{(1)}, \boldsymbol{d}_{r_2}^{(2)}, \ldots, \boldsymbol{d}_{r_n}^{(n)}]$ we get:

$$\begin{aligned}
\boldsymbol{\alpha}[\boldsymbol{d}_r, \emptyset] &= \bigoplus_{\text{all } \mathcal{F}(f)} \bigotimes_{i=1}^{n} \boldsymbol{\alpha}[\boldsymbol{d}_{r_i}^{(i)}, \mathcal{F}(f)_i] \\
&= \bigoplus_{\text{all } \rho} \bigotimes_{i=1}^{n} \delta(\rho(E_h), lab_V(v), \rho(E_t)) && \boldsymbol{d}_{r_i}^{(i)} \text{ has only 1 node } v \\
&= \bigoplus_{\text{all } \rho} \delta(\rho) \\
&= [[M]](\boldsymbol{d}_r)
\end{aligned}$$

□

Theorem 4.6. *The recognition/graph weight computation algorithm provided in [7] (and given in the appendix) is correct, i.e we return $[[M]](D)$ for input DAG D.*

Proof. Let $\boldsymbol{d}$ be the input DAG of rank $r = (0, 0)$ with n nodes.

We first show that after each edge contraction, the algorithm maintains a grounded context decomposition $\boldsymbol{d} = \boldsymbol{c}[\boldsymbol{t}_1, \ldots, \boldsymbol{t}_k]$ as well as maintaining the values of $\boldsymbol{\alpha}[\boldsymbol{t}_i, f]$ for all $\boldsymbol{t}_i$ and all possible f. We show this by induction on the number of edge contractions.

Before the first edge contraction, we have $\boldsymbol{d} = \boldsymbol{c}_1[\boldsymbol{d}_{r_1}^{(1)}, \boldsymbol{d}_{r_2}^{(2)}, \ldots, \boldsymbol{d}_{r_n}^{(n)}]$, and the first section of the algorithm iterates through all these nodes and computes all $\boldsymbol{\alpha}[\boldsymbol{d}_{r_i}^{(i)}, f]$.

For the inductive case, assume that after $m - 1$ edge contractions, we have a decomposition $\boldsymbol{d} = \boldsymbol{c}[\boldsymbol{t}_1, \ldots, \boldsymbol{t}_k]$ and values of all $\boldsymbol{\alpha}[\boldsymbol{t}_i, f]$. When we contract the next edge, observe that by design we contract an edge between two graph fragments $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ since we delete any intra-fragment edges from the contraction candidates (edges in I). By contracting this edge, the algorithm forms a 'supervertex' v corresponding to a larger graph fragment $\boldsymbol{t}' = \boldsymbol{c}'[\boldsymbol{t}_i, \boldsymbol{t}_j]$ for some context $\boldsymbol{c}'$ (implicitly defined). We then compute the 'dangling edges', denoted by $star(v)$ in the algorithm, by combining the dangling edges of $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ (excluding edges between the two fragments).

Now, by definition for a mapping $h : star(v) \to Q$, if we have mappings f, g for the dangling edges of $\boldsymbol{t}_i, \boldsymbol{t}_j$ respectively such that f, g do not conflict on shared edges ($f|_I = g|_I$) and $h = f \cup g \setminus f|_I$, the set $\{f, g\}$ is a valid $\mathcal{F}(h)$. Moreover, any $\mathcal{F}(h)$ must be expressible in this form. Using the definition of partial weight and Lemma 4.4 we get:

$$\begin{aligned}
\boldsymbol{\alpha}[\boldsymbol{c}'[\boldsymbol{t}_i, \boldsymbol{t}_j], h] &= \bigoplus_{\text{all } \mathcal{F}(h)} \boldsymbol{\alpha}[\boldsymbol{t}_i, h_i] \otimes \boldsymbol{\alpha}[\boldsymbol{t}_j, h_j] \\
&= \bigoplus_{\text{all } h = f \cup g \setminus f|_I} \boldsymbol{\alpha}[\boldsymbol{t}_i, f] \otimes \boldsymbol{\alpha}[\boldsymbol{t}_j, g]
\end{aligned}$$

This is exactly what the recognition algorithm computes.

Finally, since in each edge contraction we merge two graph fragments, after $n - 1$ edge contractions we terminate at a decomposition $\boldsymbol{d} = \boldsymbol{c}''[\boldsymbol{t}_{final}]$ and compute $\boldsymbol{\alpha}[\boldsymbol{c}''[\boldsymbol{t}], h]$, where $h = \emptyset$ as $\boldsymbol{d}$ has rank $(0, 0)$. By Lemma 4.5, this is equal to $[[M]](\boldsymbol{d})$. □

5 Determinism, Hypergraph Representation, and Equivalence of States

[6] define notions of equivalence and minimisation for top-down deterministic DAG automata, and in doing so prove that the equivalence problem is decidable in polynomial time for deterministic automata. In this section we explore these ideas and present alternative but equivalent definitions for equivalence of states by viewing a DAG automaton as a hypergraph. In this sense, we mirrors the definitions of state equivalence for tree automata.

Definition 5.1 (Deterministic Automaton)**.** *A DAG automaton $M = \langle \Sigma, Q, \Theta, \delta, \mathbb{K} \rangle$ is top down deterministic if given a symbol $\sigma \in \Sigma$, for each $\alpha \in Q^*$ we have at most one $\beta \in Q^*$ such that $\alpha \xrightarrow{\sigma} \beta$ is a rule in Θ. An automaton is bottom up deterministic if for a symbol $\sigma \in \Sigma$, for each $\beta \in Q^*$, we have at most one $\alpha \in Q^*$ such that $\alpha \xrightarrow{\sigma} \beta$ is a rule in Θ.*

In a deterministic automaton, there is at most one accepting run ρ per DAG D (but this does not necessarily hold for DAG fragments).

It the class of deterministic recognisable DAG languages, i.e the DAG languages recognised by either a bottom up or top down automaton, is a strict subset of the class of recognisable DAG languages. In other words, determinism strictly decreases the power of our automaton. The argument, presented in [6], is quite simple and we summarise the main ideas. Regular tree languages are also regular DAG languages as DAG automata generalise tree automata. Now consider a top down deterministic DAG automaton recognising a tree language (which is also a DAG language). Such a DAG automaton can also be expressed as a top down deterministic tree automaton. However, since there are regular tree languages that cannot be recognised by a top down deterministic tree automaton, such a language (which is also a regular DAG language) also cannot be recognised by a top down deterministic DAG automaton.

For the remainder of this section we will focus on top down deterministic automata. Note however that the concepts still apply to bottom up deterministic automata, as we can view a bottom up deterministic automaton as a top down one accepting the 'reverse' language (i.e the DAG edge directions are reversed). This also implies bottom up deterministic DAG automata have the same power as top down deterministic automata, and are strictly weaker than nondeterministic automata. This notion of a 'dual' automaton is delineated in [6].

Definition 5.2 (Reduced Automaton)**.** *A DAG automaton $M = \langle \Sigma, Q, \Theta, \delta, \mathbb{K} \rangle$ is reduced if there are no useless transitions. A transition $t \in \Theta$ is not useless if there is at least one DAG $D \in L(M)$ for which there exists a run ρ with $\delta(\rho) \neq 0$, such that $t = (\rho(in(v)), lab_V(v), \rho(out(v)))$ for some vertex v in G.*

For the remainder of this section we will assume DAGs are reduced. This can be performed in polynomial time as shown in [6].

Definition 5.3 (Automata Hypergraphs and Hyperpaths)**.** *Let $M = \langle \Sigma, Q, \Theta, \delta, \mathbb{K} \rangle$ be a DAG Automaton. We can view this automaton as a hypergraph, with each state $q \in Q$ forming a node, and each transition $\alpha \xrightarrow{a} \beta$ representing a hyperedge e with $src(e) = \alpha$ and $tar(e) = \beta$, where $\alpha, \beta \in Q^*$. Note that transitions are between ordered sequences of states, and therefore we may have two hyperedges/transitions between two sets of states, with the only difference being the orderings.*

*Then, we define a **hyperpath** $\boldsymbol{\pi}$ through this hypergraph as a tuple (I, F, S), where I is an initial ordered sequence of states $q_1 \dots q_m$, F is a final ordered sequence of states $p_1 \dots p_n$, and S is a sequence of hyperedges/transitions $t_1, t_2, \dots, t_k$ such that the following algorithm takes in I, S as input and returns a multiset with the same elements as F:*

Algorithm 1: HYPERPATHTRAVERSAL(I, S)

```
1 states ← {I} ;                    /* {I} is the multiset of sequence elements */
2 foreach α -a-> β ∈ S ;                          /* Iterate in sequence order */
3 do
4     states ← (states \ {α}) ∪ {β}
5 return states
```

This effectively enforces a topological order on the hyperedge sequence such that we can only pick a hyperedge in our path if its source states are 'available'.

*We define the **weight** of a hyperpath $\boldsymbol{\pi} = (I, F, S)$ as $\boldsymbol{w}(\boldsymbol{\pi}) = \bigotimes_{t \in S} \delta(t)$.*

Definition 5.4 (Yield of Hyperpath)**.** *The unique **yield** of a hyperpath (I, F, S) is the graph fragment $\boldsymbol{d} = (V, E, src, tar, lab_V, E_h, E_t)$ and run $\rho : E \cup E_h \cup E_t \to Q$ such that:*

- *$\rho(E_h) = I$, i.e each 'head' edge is assigned the corresponding state in the initial sequence (we match the i'th edge with the i'th state)*

- $\rho(E_t) = F$
- *For every* $v \in V$, $(\rho(in(v)), lab_V(v), \rho(out(v))) \in S$.

With this definition, we can see that a run ρ of automaton M on a DAG D corresponds to a hyperpath $\boldsymbol{\pi}$ in M's hypergraph with that $I = F = \lambda$ such that $\boldsymbol{\pi}$ yields G and $\boldsymbol{w}(\boldsymbol{\pi}) = \delta(\rho)$.

We now have the tools to present an alternative characterisation of equivalent states in an unweighted DAG automaton. This mirrors the state equivalence definition of tree automata, using the hypergraph representation of the automaton. We also show that this definition is equivalent to the inductive definition of [6].

Definition 5.5 (Equivalence and Distinguishability of States (Unweighted))**.** *Let* $M = \langle \Sigma, Q, \Theta \rangle$ *be a reduced top down deterministic DAG automaton (unweighted so we omit* δ *and* $\mathbb{K}$*).*

Two states $q_1, q_2 \in Q$ *are* ***equivalent*** *if the following holds: For all sequences* I_1 *with an occurrence of* q_1 *at position* i*, if there exists a hyperpath from* I_1 *to some* F_1 *yielding* $\boldsymbol{d}$ *with run* ρ_1*, then if we replace position* i *in* I_1 *with* q_2 *to form* I_2*, then there must exist a hyperpath from* I_2 *to some* F_2 *yielding* $\boldsymbol{d}$ *with run* ρ_2 *obeying the following:*

$$\rho_2(e) = \begin{cases} \rho_1(e) & \text{for } e \in E_h \text{ except the } i\text{'th edge } e_i \\ q_2 & \text{for the } i\text{'th edge in } E_h,\ e_i \end{cases}$$

For all other $e \in E \cup E_t$ *we place no restrictions on* ρ_2*.*

If the above doesn't hold, then q_1 *and* q_2 *are* ***distinguishable****.*

Theorem 5.6. *This definition of distinguishable states is equivalent to that of [6] (provided in Appendix).*

Proof. The base case of Blum's definition says that two states p, p' are distinguishable if there exists a transition $\alpha p \alpha' \xrightarrow{a} \beta$ but no transition $\alpha p' \alpha' \xrightarrow{a} \beta'$ for $\alpha, \alpha', \beta, \beta' \in Q^*$. In this case, let $I_1 = \alpha p \alpha'$, $F_1 = \beta$. Then, there exists the hyperpath containing just the hyperedge/transition $\alpha p \alpha' \xrightarrow{a} \beta$ yielding a but there does not exist a hyperpath yielding a for $I' = \alpha p' \alpha'$ since no single transition consuming symbol a has source states $\alpha p' \alpha'$.

For the inductive case, in Blum's definition, p, p' are distinguishable if there exist transitions $\alpha_0 p \alpha_0' \xrightarrow{a_0} \beta_0 q_0 \beta_0'$ and $\alpha_0 p' \alpha_0' \xrightarrow{a_0} \gamma_0 q_0' \gamma_0'$ such that q_0 and q_0' are distinguishable. 'Unrolling' this definition, we must have some sequence of transitions $\alpha_0 p \alpha_0' \xrightarrow{a_0} \beta_0 q_0 \beta_0', \alpha_1 q_0 \alpha_1' \xrightarrow{a_1} \beta_1 q_1 \beta_1', \ldots, \alpha_k q_{k-1} \alpha_k' \xrightarrow{a_k} \beta_k$ and $\alpha_0 p' \alpha_0' \xrightarrow{a_0} \gamma_0 q_0' \gamma_0', \alpha_1 q_0' \alpha_1' \xrightarrow{a_1} \gamma_1 q_1' \gamma_1', \ldots, \alpha_{k-1} q_{k-2}' \alpha_{k-1}' \xrightarrow{a_{k-1}} \gamma_{k-1} q_{k-1}' \gamma_{k-1}'$ such that there is no transition $\alpha_k q_{k-1}' \alpha_k' \xrightarrow{a_k} ...$

As such, if we let $I = \alpha_0 p \alpha_0' \alpha_1 \alpha_1' \ldots \alpha_k \alpha_k'$ and $F = \beta_0 \beta_0' \beta_1 \beta_1' \ldots \beta_{k-1} \beta_{k-1}' \beta_k$ the hyperpath formed by the first sequence of rules above yields the graph fragment $\boldsymbol{d}$, consisting of a chain of $a_0, a_1, \ldots, a_k$ (and many dangling edges). We also get the run ρ_1 which assigns states to edges based on the transitions in the hyperpath. However, if we replace p by p' in I to form I', we have no hyperpath yielding $\boldsymbol{d}$ with run ρ_2 as defined in Def 5.5. To see why, observe that as per the definition of ρ_2 and top down determinism, we must take the transition $\alpha_0 p' \alpha_0' \xrightarrow{a_0} \gamma_0 q_0' \gamma_0'$ in order to yield the first node a_0 with the same ordered sequence of edges. Similarly, we must then yield a node with label a_1 such that incoming edges from E_h must match the a_1 node in $\boldsymbol{d}$ and their states in ρ_2 must match ρ_1. Further, the j'th outgoing edge from node a_0 (assigned state q_0) is connected to an incoming edge of node a_1. This is only possible if we take the transition $\alpha_1 q_0' \alpha_1' \xrightarrow{a_1} \gamma_1 q_1' \gamma_1'$. By repeating this argument, we conclude that we must take transitions as per the second sequence above, until we reach the last transition, at which point we cannot yield a node labelled a_k since we have no transition $\alpha_k q_{k-1}' \alpha_k' \xrightarrow{a_k} ....$

We have thus shown that any two states distinguished according to Blum's definition are distinguished by our definition.

We now show the converse, i.e any states distinguished by our definition are also distinguished by that of Blum. Let q, q' be two distinguishable states such that there exists I, F with a hyperpath $\boldsymbol{\pi}$ between them yielding $\boldsymbol{d}$ and q occurs in the i'th position of I. As these states are distinguishable, if we replace the

i'th state in I with q' to create I', there is no hyperpath from I' to some F' yielding $\boldsymbol{d}$. Let $S = t_1 t_2 \dots t_k$ be the hyperpath yielding $\boldsymbol{d}$ prior to state replacement. We shall induct on the length of S.

The base case where $k = 1$ is simple, as the existence of a graph fragment $\boldsymbol{d}$ implies there is a rule $\alpha q \alpha' \xrightarrow{a} \beta$, but the nonexistence of a graph fragment containing the node a after state replacement implies there is no rule $\alpha q' \alpha' \xrightarrow{a} \dots$, aligning with the base case of Blum's definition.

We will split the inductive case into two cases. Consider the first transition $t_1 = \alpha \xrightarrow{a} \beta$. If the i'th state q is not a part of α, then replacing q with q' makes no difference to this transition. Therefore, we can remove this transition and create I_2 by removing α from I and appending β. This results in a new DAG fragment $\boldsymbol{d'}$ yielded by hyperpath $S' = t_2 \dots t_k$, such that $\boldsymbol{d'}$ cannot be yielded by replacing q with q' in I_2, and we apply the inductive hypothesis.

For the other case, we have $t_1 = \alpha q \alpha' \xrightarrow{a} \beta$. If we cannot construct $\boldsymbol{d}$ post state replacement because there is no rule $\alpha q' \alpha' \xrightarrow{a} \dots$ then we are done immediately. If on the other hand there is a rule $\alpha q' \alpha' \xrightarrow{a} \gamma$, then the state replacement has no effect on this node with label a. Therefore, we can create I_2 by removing $\alpha q \alpha'$ from I and appending β. Then, $S' = t_2 \dots t_k$ must yield a graph fragment $\boldsymbol{d'}$ similar to $\boldsymbol{d}$ but without the 'first' node labelled a. However, there must be a state $p \in \beta$ and a state $p' \in \gamma$ such that replacing p with p' then there cannot be a hyperpath yielding $\boldsymbol{d'}$ (if there was, we contradict our initial assumption that replacing q with q' makes it impossible to yield $\boldsymbol{d}$). Thus, we can apply the inductive hypothesis and therefore the statement holds for all hyperpath lengths.

We have now shown that if two states are distinguished by our definition, they are also distinguished by that of Blum's, and combining with the first part, we conclude that they are equivalent. □

We also present a definition of cyclic automata.

Definition 5.7. *(Cyclic Automaton)*

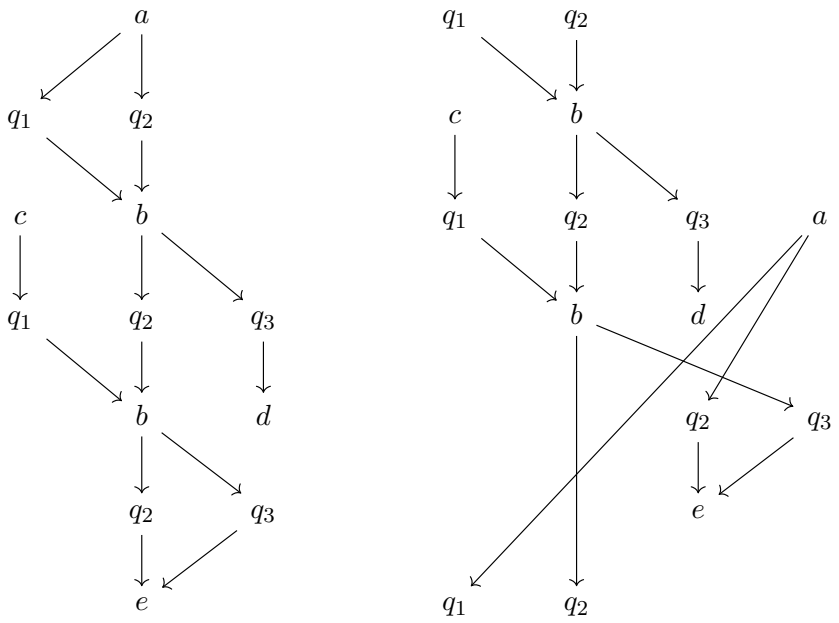
A reduced DAG automaton $M = \langle \Sigma, Q, \Theta \rangle$ is cyclic if there exists a hyperpath $\boldsymbol{\pi} = (K, K, S)$, $K \neq \lambda$, where S is a nonempty sequence of transitions such that $\boldsymbol{\pi}$ yields a connected DAG fragment. In other words, there is a sequence of transitions that takes an initial sequence of states to the same sequence of states. This can also be viewed as the hypergraph of the automaton M being cyclic.

Conjecture 5.8. *Given a reduced DAG automaton $M = \langle \Sigma, Q, \Theta \rangle$, the language of connected DAGs recognised by M, denoted $L(M)$ is infinite iff M is cyclic.*

Proof Idea. If $L(M)$ is infinite, then for some $D \in L(M)$ and a nonzero weight run ρ on D, we must have a sequence of connected transitions $t_1, t_2 \dots, t_k, t_1$. By connected transitions we mean that two consecutive transitions share an edge with the same state under ρ. This follows as $L(M)$ being infinite means that path lengths in the DAGs are unbounded, but the set of transitions is finite. Let $e_1, e_2, \dots, e_k$ be the path that connects the transitions, i.e t_1 and t_2 both use edge e_1, t_2 and t_3 both use edge e_2, and t_k and the repeat t_1 both use e_k.

We claim that for $t_i = \alpha_i \xrightarrow{a_i} \beta_i$, if we set $K = \alpha_1 \alpha_2 \dots \alpha_k \setminus \{\rho(e_1), \rho(e_2), \dots, \rho(e_k)\}$ (meaning we remove all instances of state $\rho(e_i)$), we can find an S that takes K to K, i.e we can find a hyperpath $\boldsymbol{\pi} = (K, K, S)$.

We leave this as a conjecture, and defer the proof to a future paper but the idea is to use the hyperpath that yields D (which includes the sequence of transitions $t_1, \dots, t_k$) and 'rearrange' the connections. See the following diagram for an example:



Where the left diagram is D, with a sequence of transitions $q_1q_2 \xrightarrow{b} q_2q_3, \lambda \xrightarrow{c} q_1, q_1q_2 \xrightarrow{b} q_2q_3$, and the right diagram represents the yield of a hyperpath from $K = q_1q_2$ to itself again. Observe that the transitions used are exactly the same as those of D but just rearranged.

Similarly if M is cyclic, then there exists a hyperpath $\boldsymbol{\pi} = (K, K, S)$ yielding a connected DAG fragment $\boldsymbol{d}$. We can then 'pump' this by traversing the hyperpath multiple times, as we always reach the same set of states. As $\boldsymbol{d}$ is connected, by repeatedly traversing/pumping the hyperpath k times, we yield a larger connected DAG fragment which we shall denote $\boldsymbol{d}^k$. However, this process must also have a way to terminate, i.e yield a rank $(0, 0)$ DAG fragment.

We claim that this termination process to 'tie up' the dangling edges can be done in a manner similar to the above diagram. Since M is reduced, each transition $\boldsymbol{t}_i$ is used in some DAG $\boldsymbol{d}_i$ for some nonzero weight run ρ_i. We can thus construct each of these DAGs $\boldsymbol{d}_i$ and remove the node corresponding to the transition $\boldsymbol{t}_i$, leaving us with dangling edges corresponding to the left and right hand side states of $\boldsymbol{t}_i$. Then, we claim that the set of these (currently disconnected) DAGs $\boldsymbol{d}_i$ can be connected with the DAG fragment $\boldsymbol{d}$ to yield an overall connected DAG. Hence, we can pump this infinitely to yield infinitely many connected DAGs.

Again, we leave this termination process as a conjecture to be formally proved in a future paper.

□

We believe this definition of cyclic automata is equivalent to that of [6], as assuming the above conjectures hold, we obtain the same result that M is cyclic $\iff L(M)$ is infinite.

Note also that if we allow $K = \lambda$, we arrive at the previous result where any automaton M with $L(M) \neq \emptyset$ also accepts the infinite language $L(M)^{\&}$, as we can 'pump' any connected DAG $\boldsymbol{d}$ to create infinitely many disconnected copies.

6 Algorithms for Intersection and Union

In this section we will provide algorithms (and hence constructive proofs) that the set of recognisable weighted DAG languages is closed under intersection and union. The following algorithms no longer require the automata to be reduced or deterministic.

6.1 Union

Given weighted DAG automata M_1 and M_2, we define the union of their languages $L(M_1)$ and $L(M_2)$, $L = L(M_1) \cup L(M_2)$ such that $L(D) = [[M_1]](D) \oplus [[M_2]](D)$.

The following algorithm computes the union automaton, assuming that $Q_1 \cap Q_2 = \emptyset$.

Algorithm 2: UNION$(M_1 = \langle \Sigma, Q_1, \Theta_1, \delta_1, \mathbb{K} \rangle, M_2 = \langle \Sigma, Q_2, \Theta_2, \delta_2, \mathbb{K} \rangle)$

$Q_M \leftarrow Q_1 \cup Q_2$
$\Theta_M \leftarrow \Theta_1 \cup \Theta_2$
$\delta_M \leftarrow \delta_1 \cup \delta_2$
return $\langle \Sigma, Q_M, \Theta_M, \delta_M, \mathbb{K} \rangle$

Theorem 6.1. *Given $D \in \mathcal{D}_\Sigma$, weighted DAG automata M_1 and M_2, $M =$ UNION(M_1, M_2) gives us an automaton such that $[[M]](D) = [[M_1]](D) \oplus [[M_2]](D)$.*

Proof. Since $Q_1 \cap Q_2 = \emptyset$, we also have $\Theta_1 \cap \Theta_2 = \emptyset$. This means that for any transition $\alpha \xrightarrow{a} \beta \in \Theta_M$, we have $\alpha \in Q_1^* \iff \beta \in Q_1^*$. This also means that any run ρ with $\delta_M(\rho) \neq \mathbf{0}$ must be of the form $\rho : E \to Q_1$ or $\rho : E \to Q_2$ and thus we get:

$$\begin{aligned}
[[M]](D) &= \bigoplus_{\text{all } \rho \text{ on } D} \delta_M(\rho) \\
&= \bigoplus_{\rho : E \to Q_1} \delta_M(\rho) \oplus \bigoplus_{\rho : E \to Q_2} \delta_M(\rho) \\
&= \bigoplus_{\text{all } \rho_1 \text{ on } D} \delta_1(\rho_1) \oplus \bigoplus_{\text{all } \rho_2 \text{ on } D} \delta_2(\rho_2) \\
&= [[M_1]](D) \oplus [[M_2]](D)
\end{aligned}$$

□

As the collection of $\rho : E \to Q_1$ is exactly the set of runs ρ_1 of M_1, and by definition $\delta_M(\alpha \xrightarrow{a} \beta) = \delta_1(\alpha \xrightarrow{a} \beta)$ when $\alpha, \beta \in Q_1^*$, hence $\delta_M(\rho_1) = \delta_1(\rho_1)$. The case for $\rho : E \to Q_2$ is symmetric.

6.2 Intersection

Given weighted DAG automata M_1 and M_2, we define the intersection of their languages $L(M_1), L(M_2)$, $L = L(M_1) \cap L(M_2)$ such that $L(D) = [[M_1]](D) \otimes [[M_2]](D)$.

The following algorithm computes the intersection automaton, assuming that $Q_1 \cap Q_2 = \emptyset$.

Algorithm 3: INTERSECT$(M_1 = \langle \Sigma, Q_1, \Theta_1, \delta_1, \mathbb{K} \rangle, M_2 = \langle \Sigma, Q_2, \Theta_2, \delta_2, \mathbb{K} \rangle)$

$Q_M \leftarrow Q_1 \times Q_2$
$\Theta_M \leftarrow \emptyset$
$\delta_M \leftarrow \mathbf{0}$
foreach $t_1 = q_1 q_2 \dots q_{m_1} \xrightarrow{\sigma_1} r_1 r_2 \dots r_{n_1} \in \Theta_1$ **do**
  foreach $t_2 = q'_1 q'_2 \dots q'_{m_2} \xrightarrow{\sigma_2} r'_1 r'_2 \dots r'_{n_2} \in \Theta_2$ **do**
    if $m_1 = m_2, n_1 = n_2, \sigma_1 = \sigma_2$ **then**
      $t_M = (q_1 q'_1)(q_2 q'_2) \dots (q_{m_1} q_{m_2}) \xrightarrow{\sigma_1} (r_1 r'_1)(r_2 r'_2) \dots (r_{n_1} r'_{n_2})$
      $\Theta_M \leftarrow \Theta_M \cup \{t_M\}$
      $\delta_M(t_M) = \delta_1(t_1) \otimes \delta_2(t_2)$
return $\langle \Sigma, Q_M, \Theta_M, \delta_M, \mathbb{K} \rangle$

Theorem 6.2. *Given $D \in \mathcal{D}_\Sigma$, weighted DAG automata M_1 and M_2, $M =$ INTERSECT(M_1, M_2) gives us an automaton such that $[[M]](D) = [[M_1]](D) \otimes [[M_2]](D)$.*

Proof. Let $\rho_M : E \to Q_M$ be a run of M on D. Then, $\rho_{M_1} : E \to Q_1, \rho_{M_1}(e) = \rho_M(e)[0]$ and $\rho_{M_2} : E \to Q_2, \rho_{M_2}(e) = \rho_M(e)[1]$ are runs of M_1 and M_2 on D respectively (where $\rho(e)[i]$ refers to the i'th element of the tuple indexed at 0).

Defining $t_{1v} = q_1 \dots q_{m_v} \xrightarrow{lab_V(v)} r_1 \dots r_{n_v} = \langle \rho_{M_1}(in(v)), lab_V(v), \rho_{M_1}(out(v)) \rangle$ and $t_{2v} = q'_1 \dots q_{m_v}, \xrightarrow{lab_V(v)} r'_1 \dots r'_{n_v} = \langle \rho_{M_2}(in(v)), lab_V(v), \rho_{M_2}(out(v)) \rangle$, the run weight of ρ_M is:

$$
\begin{aligned}
\delta_M(\rho_M) &= \bigotimes_{v \in V} \delta(\langle \rho_M(in(v)), lab_V(v), \rho_M(out(v)) \rangle) \\
&= \bigotimes_{v \in V} \delta(\langle (q_1 q'_1)(q_2 q'_2) \dots (q_{m_v} q'_{m_v}), lab_V(v), (r_1 r'_1)(r_2 r'_2) \dots (r_{n_v} r'_{n_v}) \rangle) \\
&= \bigotimes_{v \in V} \delta_1(t_{1v}) \otimes \delta_2(t_{2v}) && \text{Line 9} \\
&= \bigotimes_{v \in V} \delta_1(t_{1v}) \otimes \bigotimes_{v \in V} \delta_2(t_{2v}) = \delta_1(\rho_{M_1}) \otimes \delta_2(\rho_{M_2})
\end{aligned}
$$

Note that the second last line is only defined if both t_{1v} and t_{2v} are in Θ_1 and Θ_2, but if this is not the case then by definition the transition weight is 0. More formally, if one of $t_{iv} \notin \Theta_i$, then $\delta_i(\rho_{M_i}) = 0$ and $\delta_M(\rho_M) = 0$.

Hence, every run ρ_M of M on D has associated with it two runs, ρ_{M_1} of M_1 and ρ_{M_2} of M_2, such that we can write $\delta_M(\rho_M) = \delta_1(\rho_{M_1}) \otimes \delta_2(\rho_{M_2})$

Thus, we have:

$$
\begin{aligned}
[[M]](D) &= \bigoplus_{\text{all } \rho \text{ on } D} \delta_M(\rho) \\
&= \bigoplus_{\rho_{M_1}, \rho_{M_2} \text{ on } D} \delta_1(\rho_{M_1}) \otimes \delta_2(\rho_{M_2}) \\
&= \bigoplus_{\rho_{M_1} \text{ on } D} \delta_1(\rho_{M_1}) \otimes \bigoplus_{\rho_{M_2} \text{ on } D} \delta_2(\rho_{M_2}) && \text{by distributivity} \\
&= [[M_1]](D) \otimes [[M_2]](D)
\end{aligned}
$$

□

7 Conclusions and Future Work

In this paper we have examined properties of DAG automata, providing definitions and properties in line with those of string and tree automata. Specifically, we have given a more formal definition of a partial weight and graph contexts, and thus proved correctness of the recognition algorithm presented in [7]. We then provided an alternative definition of unweighted state equivalence in terms of the hypergraph representation of an automaton, which may be a useful perspective from which to derive a definition of weighted equivalence (e.g by comparing the weights of the generated graph fragments for two equivalent states). We also provide a definition of cyclic automata, again based on the hypergraph representation and a conjecture that an automaton being cyclic means its language of connected DAGs is infinite. Finally, we provided algorithmic proofs of closure under weighted union and intersection, a simple generalisation of the techniques used for weighted string and tree automata.

We believe that there is significant potential for future work in this area. For instance, concepts of weighted equivalence, Myhill-Nerode theorems, and weight pushing can be investigated. Hopefully, this would lead to algorithms for weighted determinisation and weighted minimisation.

Training algorithms for weighted DAG automata also represent a valuable area of research, given the existence of training/weight learning algorithms for string and tree automata.

Graph grammars are also a well studied topic, gaining traction after the seminal paper [8] introducting hyperedge replacement graph grammars. The connection between graph grammars (perhaps the node

replacement variant as described in [13]) and DAG automata may also be worth exploring, especially the relationship with our definition of graph contexts.

References

[1] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

[2] Henrik Björklund, Johanna Björklund, and Petter Ericson. 2017. On the regularity and learnability of ordered dag languages. In *Implementation and Application of Automata: 22nd International Conference, CIAA 2017, Marne-la-Vallée, France, June 27-30, 2017, Proceedings 22*, pages 27–39. Springer.

[3] Henrik Björklund, Johanna Björklund, and Petter Ericson. 2018. *Minimisation and Characterisation of Order-Preserving DAG Grammars*. Umea University.

[4] Henrik Björklund, Frank Drewes, and Petter Ericson. 2016. Between a rock and a hard place–uniform parsing for hyperedge replacement dag grammars. In *Language and Automata Theory and Applications: 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings 10*, pages 521–532. Springer.

[5] Henrik Björklund, Frank Drewes, and Petter Ericson. 2019. Parsing weighted order-preserving hyperedge replacement grammars. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 1–11, Toronto, Canada. Association for Computational Linguistics.

[6] Johannes Blum and Frank Drewes. 2019. Language theoretic properties of regular dag languages. *Information and Computation*, 265:57–76.

[7] David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez, and Giorgio Satta. 2018. Weighted DAG automata for semantic graphs. *Comput. Linguistics*, 44(1).

[8] Frank Drewes, H-J Kreowski, and Annegret Habel. 1997. Hyperedge replacement graph grammars. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 95–162. World Scientific.

[9] Petter Ericson. 2019. *Order-preserving graph grammars*. Ph.D. thesis, Umea University, Sweden.

[10] Tsutomu Kamimura and Giora Slutzki. 1981. Parallel and two-way automata on directed ordered acyclic graphs. *Inf. Control.*, 49(1):10–51.

[11] Lutz Priese. 2007. Finite automata on unranked and unordered dags. In *Developments in Language Theory, 11th International Conference, DLT 2007, Turku, Finland, July 3-6, 2007, Proceedings*, volume 4588 of *Lecture Notes in Computer Science*, pages 346–360. Springer.

[12] Daniel Quernheim and Kevin Knight. 2012. DAGGER: A toolkit for automata on directed acyclic graphs. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing, FSMNLP 2012, Donostia-San Sebastiían, Spain, July 23-25, 2012*, pages 40–44. The Association for Computer Linguistics.

[13] Grzegorz Rozenberg. 1997. *Handbook of graph grammars and computing by graph transformation*, volume 1. World scientific.

A Appendix: External Algorithms and Definitions

A.1 Recognition/Graph Weight Computation Algorithm

We present this algorithm as specified in [7]. $star(v)$ represents the set of incoming and outgoing edges from (super)vertex v. $f|_I$ is the function f restricted to edges in set I.

Algorithm 4: $[[M]](D)$ Computation

foreach *node* v **do**
  forall $f : star(v) \to Q$ **do**
    $\boldsymbol{\alpha}[v, f] \leftarrow \delta(\langle f[in(v)], lab(v), f[out(v)]\rangle)$
foreach *edge* e *in order, s.t.* e *has not been deleted* **do**
  $(u, v) \leftarrow (src(e), tar(e))$
  $I \leftarrow star(u) \cap star(v)$
  Create new supervertex z
  $in(z) \leftarrow in(u) \cup in(v) \setminus I$
  $out(z) \leftarrow out(u) \cup out(v) \setminus I$
  forall $h : star(z) \to Q$ **do**
    $\boldsymbol{\alpha}[z, h] \leftarrow 0$
  forall $f : star(u) \to Q$ **do**
    forall $g : star(v) \to Q$ *s.t.* $f|_I = g|_I$ **do**
      $h = f \cup g \setminus f|_I$
      $\boldsymbol{\alpha}[z, h] \leftarrow \boldsymbol{\alpha}[z, h] \oplus \boldsymbol{\alpha}[u, f] \otimes \boldsymbol{\alpha}[v, g]$
  Delete u, v and all edges in I
One supervertex v remains
return $\boldsymbol{\alpha}[v, \emptyset]$

A.2 State Equivalence

We present the definition of equivalent/distinguishable states as specified in [6].

Definition A.1 (Distinguishable State Pair)**.** *Let $A = (Q, \Sigma, \Theta)$ be a reduced top down deterministic DAG automaton. The set of all distinguishable state pairs is defined inductively as the smallest symmetric binary relation $\Delta_A \subseteq Q^2$ such that $(p, p') \in \Delta_A$ if there exists $\sigma \in \Sigma$ and $\alpha, \alpha' \in Q^*$ such that one of the following hold:*

1. *Θ contains a rule $\alpha p \alpha' \xrightarrow{\sigma} \ldots$ but no rule $\alpha p' \alpha' \xrightarrow{\sigma} \ldots$*
2. *There are rules $\alpha p \alpha' \xrightarrow{\sigma} q_1 \cdots q_n$ and $\alpha p' \alpha' \xrightarrow{\sigma} q'_1 \cdots q'_n$ in Θ such that $(q_j, q'_j) \in \Delta_A$ for some $j \in [n]$.*

Definition A.2 (Equivalent States)**.** *States $p, p' \in Q$ are equivalent, $p \equiv p'$ if $(p, p') \notin \Delta_A$.*

B Appendix: Unfinished Work (DAG Grammars)

B.1 Weighted Order Preserving DAG Grammars

B.1.1 Graph Definition

Definition B.1 (Hyperedge Labelled, Directed Hypergraph)**.** *Let $(\Sigma, rank)$ be a ranked alphabet. A directed, hyperedge labeled hypergraph over Σ is a graph $G = (V, E, src, tar)$ alongside a labelling function lab_E. $lab_E : E \to \Sigma$ is a labelling of the edges such that $rank(lab(e)) = |tar(e)|$ for $e \in E$, i.e the rank of an edge's label corresponds to the number of targets. We will express such graphs as tuples (V, E, src, tar, lab_E).*

Let $\mathcal{A}_\Sigma$ be the set of all hyperedge labelled DAGs over ranked alphabet Σ.

Definition B.2 (Marked Hypergraph)**.** *A marked hyperedge labelled, directed hypergraph is a tuple $G = (V, E, src, tar, lab_E, X)$ where (V, E, src, tar, lab_E) forms a graph as in Definition B.1 and $X \in V^{\circledast}$ is a nonempty sequence called the marking of G, and whose nodes are referred to as external nodes. As X is nonempty, we can write it as $X = (v, w)$ where $v \in V$ and $w \in V^{\circledast}$ and say that $v = root(G)$, $w = ext(G)$.*

B.1.2 Preliminaries

Let $D = (V, E, src, tar, lab_E)$ be a DAG as per Definition B.1, and $u, v \in V$. An edge e is a *common ancestor edge* of u and v if there exists $t, t' \in tar(e)$ such that u is a descendant of t and v is a descendant of t'. An edge e is the *closest common ancestor edge* of u and v if there is no other common ancestor edge e' such that $src(e') \in tar(e)$. Note: a pair of nodes may have more than one closest common ancestor edge in a DAG.

Definition B.3 (Partial Order on Leaves of DAG)**.** *Let $D = (V, E, src, tar, lab_E)$ be a DAG. We say that $u \preceq_D v$ for $u, v \in V$ if for every closest common ancestor edge e of u and v, the sequence $tar(e)$ can be written as wtw' where $w, w' \in V^{\circledast}$ and $t \in V$ such that t is an ancestor of u and any ancestors of v are in tw'.*

Claim B.4. *For DAG D, the transitive closure of $\preceq_D$ forms a partial order on the leaves of D.*

Proof. The relation is reflexive as the closest common ancestor edges of a node u and itself are the set of incoming edges to u, and the targets of such an edge e can be written wuw'. To show antisymmetry, assume that $u \preceq_D u'$ and $u' \preceq_D u$ for $u, u' \in V$. Then, it must be the case that for any closest common ancestor edge e, $tar(e) = w_1 t w_1' = w_2 t' w_2'$, where t, t' are ancestors of u, u' respectively, and $t' \in tw_1'$ and $t \in t'w_2'$. However, this is only possible if $t = t'$, and since e by assumption is a closest common ancestor, $t = t' \iff u = u'$.

Note that we need to take the transitive closure as $\preceq_D$ is not necessarily transitive on its own - consider $u \preceq_D u'$, $u' \preceq_D u''$ for $u, u', u'' \in V$ but u and u'' do not share a common ancestor edge. □

B.1.3 Definition

A (weighted) order preserving DAG grammar is a structure $H = (\Sigma, N, P, S, w, \mathbb{K})$ where:

- Σ is a ranked alphabet of terminal symbols
- N is a ranked alphabet of nonterminal symbols
- P is a set of production rules
- $S \in N$ is the starting nonterminal. We define $S^{\bullet}$ as the starting graph with a single edge labelled by S connecting a marked source node v_0 to $k = rank(S)$ marked targets $v_1, v_2, \ldots v_k$.
- $\mathbb{K}$ is a commutative semiring
- $w : P \to \mathbb{K}$ is a weight function

The rules in P must be of the form $A \to F$ where $A \in N$ and F is a marked DAG with $|ext(F)| = rank(A)$ that obeys one of the following forms:

1. F contains just two edges e_1, e_2 both labelled with A, $src(e_1) = src(e_2)$, $tar(e_1) = tar(e_2)$. Further, all nodes are marked ($V = \{x \in X\}$) and connected to the two edges. This is called a *clone* rule.
2. F has height at most 2, and obeys the following:
   - All nodes have out degree at most one.
   - There is a single root node, which is the marked $v = root(F)$, and the single edge e with $src(e) = v$ is labelled by a terminal symbol.
   - Every other edge is labelled with a nonterminal symbol.
   - Only leaves have in-degrees larger than one.
   - All targets of nonterminal edges have either in-degree larger than one or are marked nodes.
   - $\preceq_F$ gives a total ordering of the leaves and $ext(F)$ respects this ordering (i.e the sequence $ext(F)$ is in 'increasing' order with respect to $\preceq_F$).

These restrictions are harsh but are ensure polynomial parsing. Björklund et al. show how relaxing some of these restrictions leads to an NP-Complete membership problem, but a full characterisation of strictly necessary restrictions for polynomial parsing remains an open issue.

A derivation step of H on graph G, given rule $\rho = A \to F$, takes an edge $e \in E_G$ labelled with A and replaces it with the marked DAG F. $src(e)$ is identified with $root(F)$, and the length $k = rank(A)$ sequence $tar(e)$ is identified with the sequence $ext(F)$ (i'th node of $tar(e)$ is i'th node of $ext(F)$). If G' is the resultant graph, we say $G \Rightarrow_\rho G'$. We write $\Rightarrow_H^*$ to denote the reflexive transitive closure of $\Rightarrow$, such that $G \Rightarrow_H^* G'$ if G' can be derived in zero or more derivation steps using rules of H.

We denote by $L(H)$ the **language** of grammar H, where $L(H) = \{g \in \mathcal{A}_\Sigma \mid S^\bullet \Rightarrow_H^* g\}$, i.e the set of terminal labelled graphs generated from the starting graph.

In the weighted case, we also overload the weight function to define the **derivation weight** of a derivation $\boldsymbol{t} = S^\bullet \Rightarrow_{p_0} G_0 \Rightarrow_{p_1} G_1 \ldots \Rightarrow_{p_k} G_k$ as:

$$w(\boldsymbol{t}) = \bigotimes_i w(p_i)$$

The **graph weight** of $D \in \mathcal{A}_\Sigma$ under WOPDG H, denoted by $\mathcal{S}_H$ is:

$$\mathcal{S}_H(D) = \bigoplus_{\text{all distinct } \boldsymbol{t}} w(\boldsymbol{t})$$

Where two derivations are distinct if their independent derivation steps cannot be reordered to be the same. Two derivation steps p_1, p_2 are independent if for G, G' such that $G \Rightarrow_{p_1} G_1 \Rightarrow_{p_2} G'$, we also have $G \Rightarrow_{p_2} G_2 \Rightarrow_{p_1} G'$.