Towards Neuro-Symbolic Tree and Graph Transformation – an Invitation

WATA 2023, Leipzig

Frank Drewes
Joint work with Johanna Björklund, Anna Jonsson, and Yannick Stade

Contents

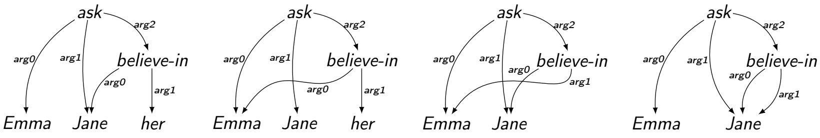
Graph Expansion Grammars

Motivation

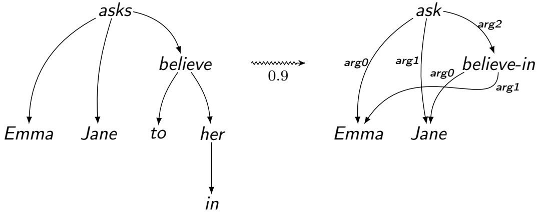
- We have been trying for a long time now to find "the right" formalism for recognizing/generating semantic graphs in Natural Language Processing.
- Desiderata:
  (1) can enforce local structural requirements
  (2) can disregard structure where necessary
  (3) has nice algorithmic properties
- Graph automata tend to fail at either (1) or (3).
- Hyperedge-replacement graph grammars are good at (1) and reasonably so at (3) but fail at (2).
- Graph expansion generalizes hyperedge replacement to support (2).

Illustrating example

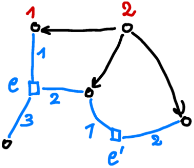
Emma asks Jane to believe in her.



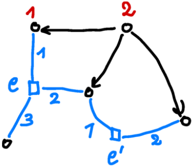
(correct) *(incorrect)* *(more likely?)* *(incorrect?)*

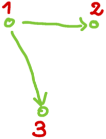


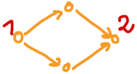
From hyperedge replacement to graph expansion (1)

Hypergraph: 

From hyperedge replacement to graph expansion (1)

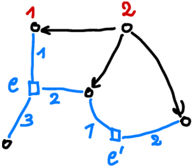
Hypergraph: 

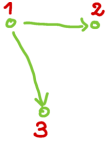
$G_1 =$ 

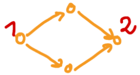
$G_2 =$ 

Constructing $H^{e,e'}(G_1, G_2)$:

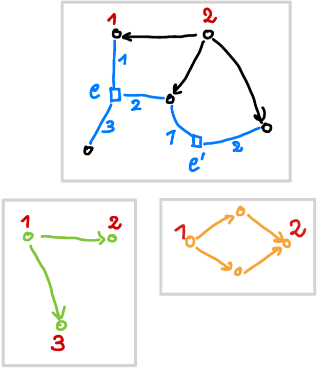
From hyperedge replacement to graph expansion (1)

Hypergraph: 

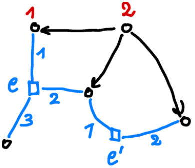
$G_1 =$ 

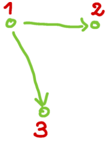
$G_2 =$ 

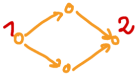
Constructing $H^{e,e'}(G_1, G_2)$:

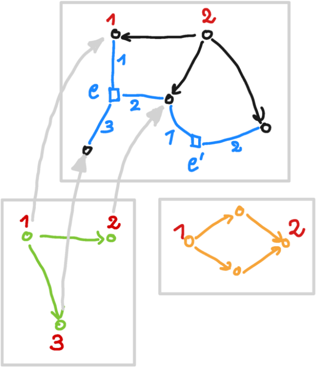


From hyperedge replacement to graph expansion (1)

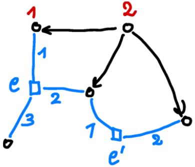
Hypergraph: 

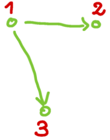
$G_1 =$ 

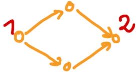
$G_2 =$ 

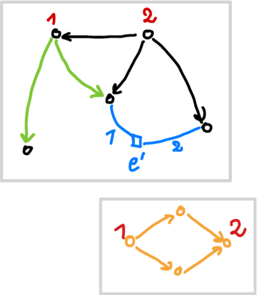
Constructing $H^{e,e'}(G_1, G_2)$: 

From hyperedge replacement to graph expansion (1)

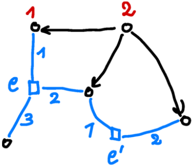
Hypergraph: 

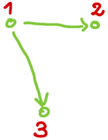
$G_1 =$ 

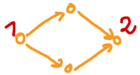
$G_2 =$ 

Constructing $H^{e,e'}(G_1, G_2)$: 

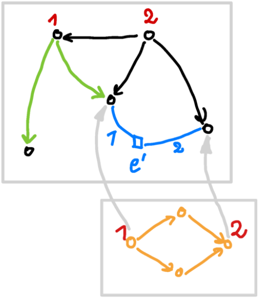
From hyperedge replacement to graph expansion (1)

Hypergraph: 

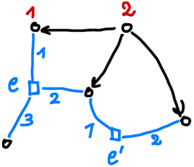
$G_1 =$ 

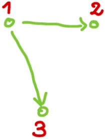
$G_2 =$ 

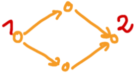
Constructing $H^{e,e'}(G_1, G_2)$:



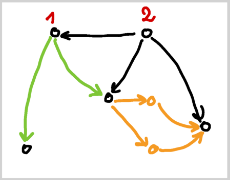
From hyperedge replacement to graph expansion (1)

Hypergraph: 

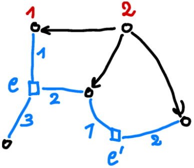
$G_1 =$ 

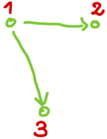
$G_2 =$ 

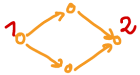
Constructing $H^{e,e'}(G_1, G_2)$:



From hyperedge replacement to graph expansion (1)

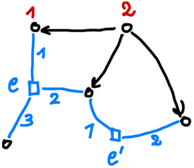
Hypergraph: 

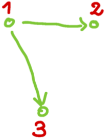
$G_1 =$ 

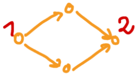
$G_2 =$ 

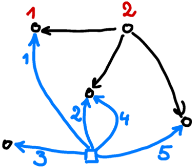
Simpler primitives:

From hyperedge replacement to graph expansion (1)

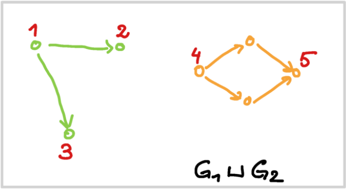
Hypergraph: 

$G_1 =$ 

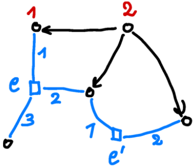
$G_2 =$ 

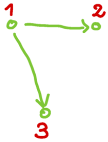


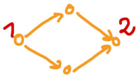
Simpler primitives:

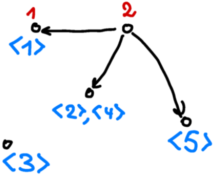


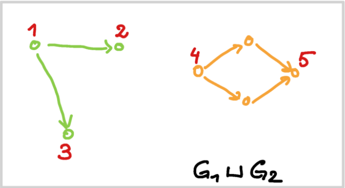
From hyperedge replacement to graph expansion (1)

Hypergraph: 

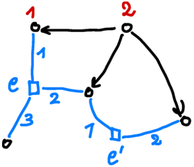
$G_1 =$ 

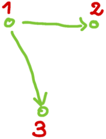
$G_2 =$ 

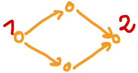


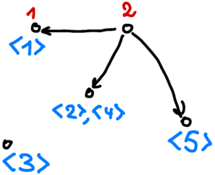
Simpler primitives: 

From hyperedge replacement to graph expansion (1)

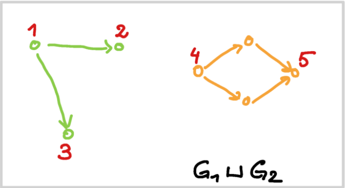
Hypergraph: 

$G_1 =$ 

$G_2 =$ 



i	$=$	i-th port
$\langle j \rangle$	$=$	j-th dock

Simpler primitives: 

From hyperedge replacement to graph expansion (2)

Graph expansion operations have additional context nodes:

- Nodes which are neither ports nor docks are context nodes.
- When the operation is applied, context nodes are identified with nondeterministically chosen nodes in the argument graph.
- To make this useful, we need a way to limit which nodes a context node can be identified with.
- In a recent paper, we proposed to use formulas φ of counting monadic second-order (CMSO) logic for this purpose.
- The context nodes $x_1, \ldots, x_k$ are the free variables of φ.
- x_i may be identified with v_i if $G \models \varphi(x_1/v_1, \ldots, x_k/v_k)$.

$\Rightarrow$ Graph expansion operations are nondeterministic!

From hyperedge replacement to graph expansion (3)

Definition: Graph Expansion Grammar

A graph expansion grammar is a pair $\Gamma = (g, \mathcal{A})$ where

- g is a regular tree grammar over a ranked alphabet Σ and
- $\mathcal{A}$ is a Σ-algebra which interprets every symbol of Σ as an expansion operation, $\sqcup$, or the empty graph $\varnothing$.[1]

The graph language generated by G is $L(\Gamma) = \bigcup_{t \in L(g)} \mathrm{val}_{\mathcal{A}}(t)$.

[1] The domain of $\mathcal{A}$ is the powerset of the set of graphs.

An Example

Trees with shortcuts to leaves (1)

An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

Trees with shortcuts to leaves (1)

An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

$S \rightarrow$ 

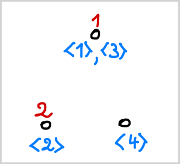
$(\varnothing)$

Trees with shortcuts to leaves (1)

An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

$S \rightarrow$ 

$(\varnothing)$

$S \rightarrow$ 

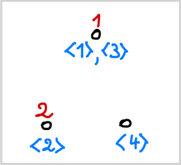
$(S \sqcup S)$

Trees with shortcuts to leaves (1)

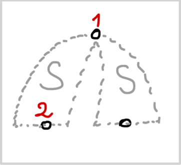
An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

$S \rightarrow$ 

$(\varnothing)$

$S \rightarrow$ 

$(S \sqcup S)$

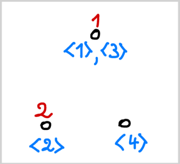
effect: 

Trees with shortcuts to leaves (1)

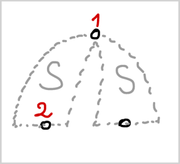
An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

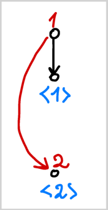
$S \rightarrow$ 

$(\varnothing)$

$S \rightarrow$ 

$(S \sqcup S)$

effect: 

$S \rightarrow$ 

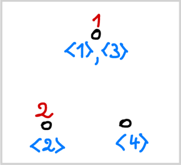
(S)

Trees with shortcuts to leaves (1)

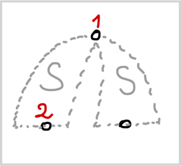
An HR grammar for trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

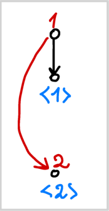
$S \rightarrow$ 

$(\varnothing)$

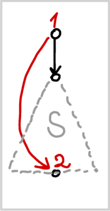
$S \rightarrow$ 

$(S \sqcup S)$

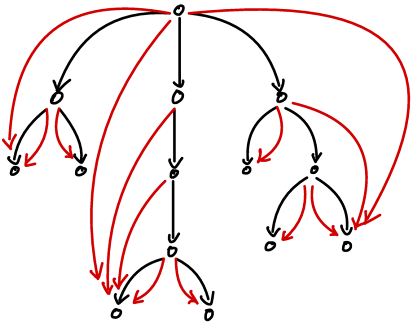
effect: 

$S \rightarrow$ 

(S)

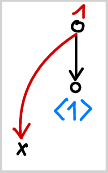
effect: 

Example tree generated:



A graph expansion grammar for all trees in which each node has a "shortcut" to a leaf in each of its direct subtrees.

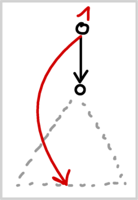
$S \quad \rightarrow$



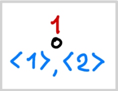
(S)

where $\neg\exists y.\mathrm{edg}(x,y)$

effect:

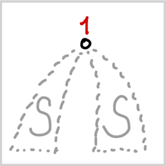


$S \quad \rightarrow$



$(S \sqcup S)$

effect:

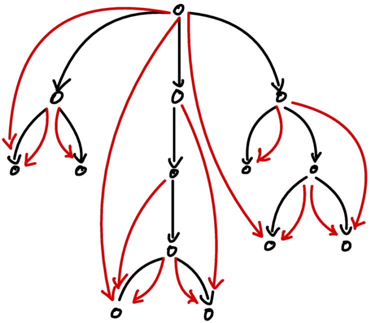


S

$(\varnothing)$

Trees with shortcuts to leaves (4)

Example tree generated:



Some Results

Observation

Graph expansion is more powerful than both CMSO and HR.

Proof. HR is included by definition. The rules

$S_0 \rightarrow \varnothing(S)$ where φ

$S \rightarrow$  (S) where true $\quad | \quad S \sqcup S \quad | \quad$ 

generate all graphs satisfying φ.

Now, take the union of a graph language in $\mathrm{CMSO} \setminus \mathrm{HR}$ with one in $\mathrm{HR} \setminus \mathrm{CMSO}$ . . .

Polynomial parsing

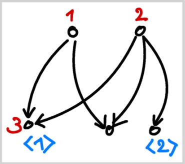
Harnessing the power of graph expansion: graph extension with local conditions

Extension operations are restricted expansion operations:

(1) Edges only from new to old nodes.

(2) All non-ports have incoming edges.

(3) Docks are pairwise distinct.



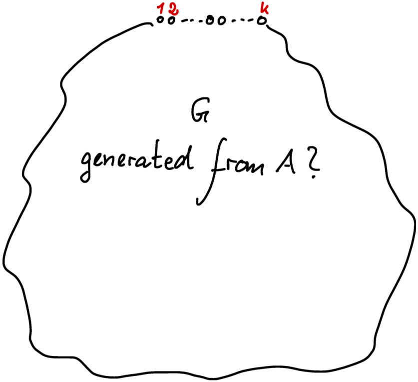
Local CMSO conditions:

(1) No direct use of the edge predicate.

(2) Instead "local" node predicates $\pi(x)$ like
for all edges (x, y) labeled a there is an edge (y, z) labeled b such that there is no edge from y to z labeled c.

Idea of the dynamic programming parsing algorithm

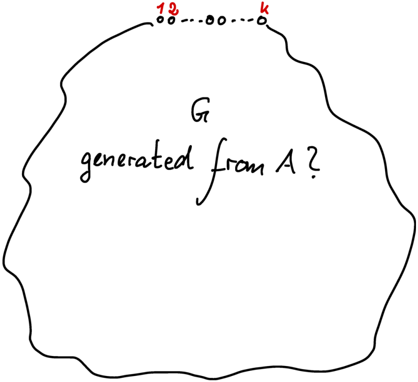
Can this graph be generated from nonterminal A?



Idea of the dynamic programming parsing algorithm

Can this graph be generated from nonterminal A?

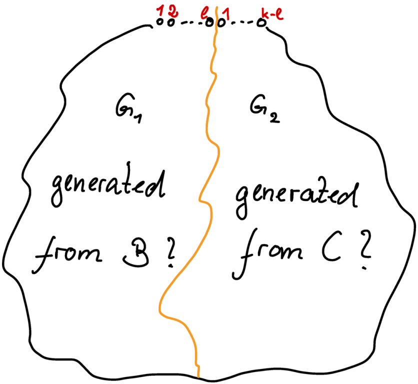
Case 1: $A \to B \sqcup C$.



Idea of the dynamic programming parsing algorithm

Can this graph be generated from nonterminal A?

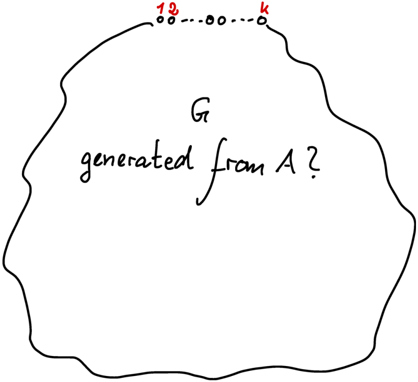
Case 1: $A \to B \sqcup C$.



Idea of the dynamic programming parsing algorithm

Can this graph be generated from nonterminal A?

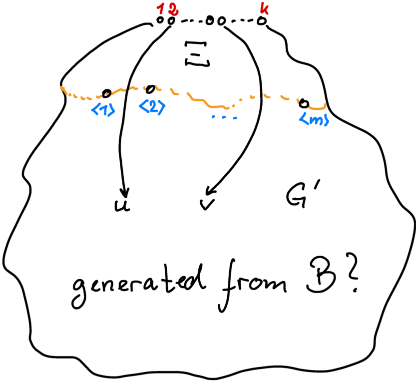
Case 2: $A \to \Xi(B)$ where φ.



Idea of the dynamic programming parsing algorithm

Can this graph be generated from nonterminal A?

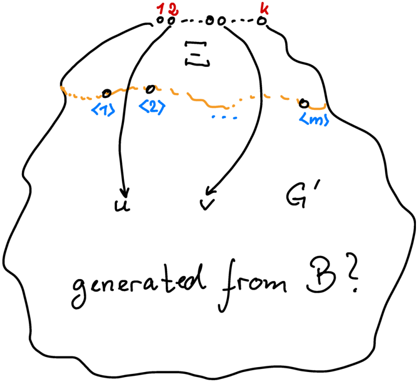
Case 2: $A \to \Xi(B)$ where φ.



Idea of the dynamic programming parsing algorithm

Can this graph be generated from nonterminal A?

Case 2: $A \to \Xi(B)$ where φ.



Additionally: is $\varphi(u, v)$ true in G?

More results about graph extension

Some additional results

- "Edge-agnostic" graph extension grammars admit a pumping lemma, and their Parikh image is semi-linear.
- This does not hold for graph extension grammars with local conditions.
- In fact, those can (in a weak sense) simulate Turing machines.

Using Weighted MSO Logic

Beware! Uncooked material ahead. . .

Informally recalling (one type of) weighted MSO logic

Prerequisite: a commutative semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$.

Weighted MSO logic (à la Droste & Dück 2015) has

- ordinary MSO formulas evaluating to 0 (false) or 1 (true),
- formulas $\varphi \odot \varphi'$ summing up/multiplying the weights of φ and φ' using $\odot \in \{\oplus, \otimes\}$,
- formulas $\bigodot_x \varphi$ summing up/multiplying the weights of $\varphi(x/v)$ over all nodes v, where $\odot \in \{\oplus, \otimes\}$, and
- formulas $\bigoplus_X \varphi$ summing up the weights of $\varphi(X/V)$ for all node sets V by applying $\oplus$.

Note: we could add $\bigotimes_X$, or may want to exclude $\bigoplus_X$ as well.

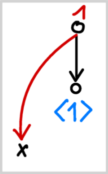
A proposal for weighted graph expansion grammars

- Replace the CMSO conditions by weighted MSO formulas.
- Applying an expansion operation Ξ to an argument graph G then yields weighted graphs, i.e., $\Xi(G)\colon \mathbb{G} \to \mathbb{S}$:
  Every assignment α of context nodes of Ξ to nodes in G determines a weighted graph $(\Xi_\alpha(G), \varphi(G,\alpha))$. We let
  $$\Xi(G)(G') = \bigoplus\{\varphi(G,\alpha) \mid \alpha \in \mathrm{A},\ \Xi_\alpha(G) \simeq G'\}.$$
- For weighted input graphs (G, w), let $\Xi(G,w) = w \otimes \Xi(G)$.
- We let $\sqcup$ translate to $\otimes$ or $\oplus$ and assign the weight 0 to $\varnothing$.
- With this, weighted graph expansion algebras $\mathcal{A}$ recursively evaluate each tree to a (finite) weighted graph language.
- Now, $L(\Gamma) = \bigoplus_{t \in L(g)} \mathrm{val}_{\mathcal{A}}(t)$.

Note: In general, the last item requires infinite sums to be defined.

An example

We want the weight to be the *length of the longest shortcut*, using $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$.

$S \to$ 

(S) where φ

We define φ as follows:

- $\text{path}(X, p, x) = $ "X is a path from p to x, which is a leaf"
- $\text{len}(p, x) = \text{MAX}_X \max(\text{path}(X, p, x), |X|)$ where $|X| = \sum_z \max(z \notin X, 1 + (z \in X))$
- $\varphi(x) = \text{MAX}_p(\text{port}_1(p) + \text{len}(p, x))$

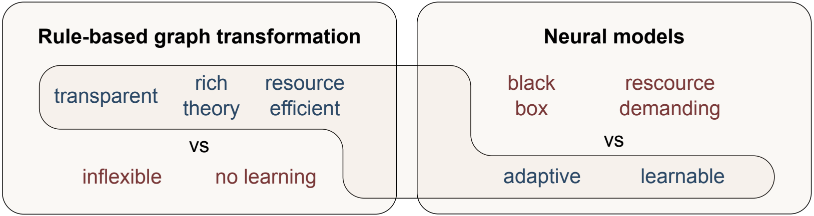
Questions to ponder

- Is the use of non-idempotent (or non-extremal) addition operators meaningful?
- In the previous example, the weight of a graph in the support of $\mathrm{val}_{\mathcal{A}}(t)$ does not depend on t. This seems to be a useful property. Can it be decided/guaranteed?
- When can we efficiently compute the weight of a graph?
- In particular, can the parsing algorithm for the unweighted case be "made weighted"?
- How to characterize the generated weighted graph languages?
- Are there equivalent (graph) automata models?

Neuro-Symbolic Graph Expansion Grammars

Beware! Raw meat ahead...

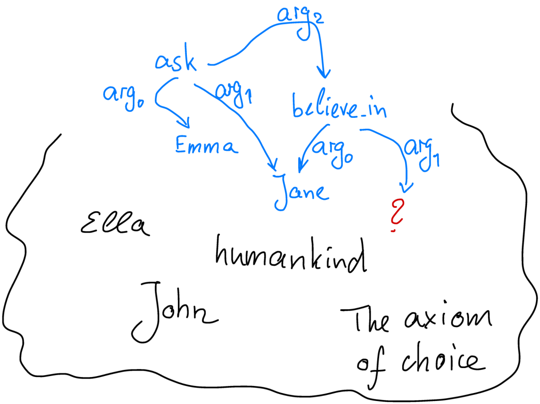
Why “neuro-symbolic” and what does it mean?



- Neuro-symbolic systems combine neural and symbolic components.
- Often, the neural component produces “input” to a symbolic system.
- I am more interested in having neural components inside the symbolic one.

The case of graph expansion grammars

Coreference resolution in natural language processing:



We may want to train a neural network to pick suitable targets!

Questions

- Can we train a neural network to pick reasonable targets for context nodes?
- If so, how? For which type of neural network? Using which kind of training data?
- How to incorporate and make use of context information, including other modalities (images, input strings, etc)?

Finally, something entirely different

Finally, something entirely different

We (Johanna, Henrik Björklund, and I) have funding for a postdoc/PhD position in this broad area. If you know interested and capable candidates, let us know.