Text Generation from Discourse Representation Structures

Citation for published version:
Liu, J, Cohen, SB & Lapata, M 2021, Text Generation from Discourse Representation Structures. in Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, Online, pp. 397-415, 2021 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 6/06/21. https://doi.org/10.18653/v1/2021.naacl-main.35

Digital Object Identifier (DOI):
10.18653/v1/2021.naacl-main.35

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Text Generation from Discourse Representation Structures

Jiangming Liu  Shay B. Cohen  Mirella Lapata
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
jiangming.liu@ed.ac.uk, {scohen,mlap}@inf.ed.ac.uk

Abstract

We propose neural models to generate text from formal meaning representations based on Discourse Representation Structures (DRSs). DRSs are document-level representations which encode rich semantic detail pertaining to rhetorical relations, presupposition, and co-reference within and across sentences. We formalize the task of neural DRS-to-text generation and provide modeling solutions for the problems of condition ordering and variable naming which render generation from DRSs non-trivial. Our generator relies on a novel sibling treeLSTM model which is able to accurately represent DRS structures and is more generally suited to trees with wide branches. We achieve competitive performance (59.48 BLEU) on the GMB benchmark against several strong baselines.

1 Introduction

It is not uncommon for text generation systems to produce natural language output from intermediate semantic representations (Yao et al., 2012; Takase et al., 2016). The literature presents several examples of generating text from logical forms underlying various grammar formalisms (Wang, 1980; Shieber et al., 1990; Carroll and Oepen, 2005; White et al., 2007), typed lambda calculus (Lu and Ng, 2011), Abstract Meaning Representations (AMR; Flanigan et al. 2016; Konstas et al. 2017; Song et al. 2018; Beck et al. 2018; Damonte and Cohen 2019; Ribeiro et al. 2019; Zhu et al. 2019; Cai and Lam 2020; Wang et al. 2020), Discourse Representation Theory (DRT; Basile and Bos 2011; Basile 2015), and Minimal Recursion Semantics (MRS; Horvat et al. 2015; Hajdik et al. 2019).

In this work, we propose neural models to generate high-quality text from semantic representations based on Discourse Representation Structures (DRSs). DRSs are the basic meaning-carrying units in Discourse Representation Theory (DRT; Kamp 1981; Kamp and Reyle 1993; Asher and Lascarides 2003), a formal semantic theory designed to handle a variety of linguistic phenomena, including anaphora, presuppositions (Van der Sandt, 1992; Venhuizen et al., 2018), and temporal expressions within and across sentences. DRSs are scoped meaning representations, they capture the semantics of negation, modals, and quantification.

Figure 1 displays in box format the meaning representation for a discourse consisting of two sentences. The outermost box is a segmented DRS expressing the rhetorical relation CONTRAST between box $b_1$ representing the first sentence and box $b_4$ representing the second sentence. Boxes $b_1$ and $b_2$ are DRSs, the top layers contain variables (e.g., $x_1, x_2$) indicating discourse referents and the bottom layers contain conditions (e.g., $\text{Name}(x_3, \text{“tom”})$) representing information about discourse referents. Variables and conditions have pointers (denoted by $b$ in the figure) pointing to the boxes where they should be interpreted. $^1$ Predicates are disambiguated to their Wordnet (Fellbaum, 1998) senses (e.g., male.n.02 and play.v.03).

Although there has been considerable activity recently in developing models which analyze text in the style of DRT (van Noord et al., 2018, 2019; Liu et al., 2019a, 2018; Fancellu et al., 2019), attempts

\begin{figure}
\centering
\begin{tabular}{|c|c|}
\hline
$\text{temp} \leftarrow \text{after}(e_1, t_1)$ & $\text{temp} \leftarrow \text{before}(e_2, e_1)$ \\
\hline
$\text{Theme}(e_1, x_2)$ & $\text{Theme}(e_2, x_2)$ \\
\hline
$\text{Agent}(e_1, \text{male.n.02})$ & $\text{Agent}(e_2, \text{male.n.02})$ \\
\hline
$\text{Pred}(e_1, \text{temp.n.01})$ & $\text{Pred}(e_2, \text{stop.v.05})$ \\
\hline
$\text{Named}(x_2, \text{“tom”})$ & $\text{Named}(x_2, \text{“tom”})$ \\
\hline
\end{tabular}
\caption{DRS representing two-sentence discourse.}
\end{figure}
to generate text from DRSs have been few and far between (however see Basile 2015 and Narayan and Gardent 2014 for notable exceptions). This is primarily due to two properties of DRS-based semantic representations which render generation from them challenging. Firstly, DRS conditions are unordered representing a set (rather than a list).\footnote{An exception are conditions in segmented DRSs whose order can be retrieved deterministically based on the arguments of rhetorical relations. For example, given the relation \textsc{BECAUSE}(b_1, b_3), we can assume that box \textit{b}_1 precedes \textit{b}_3.}

A hypothetical generator would have to produce the same output text for any DRSs which convey the same meaning but appear different due to their conditions having a different order (see Figures 1 and 2a which are otherwise identical but the order of conditions in boxes \textit{b}_1 and \textit{b}_4 varies). The second challenge concerns variables and their prominent status in DRSs. Variables identify objects in discourse (such as entities and predicates), and are commonly used to model semantic phenomena including coreference, control constructions, and scope. In Figure 1, variables \textit{x}, e, s, t, p, and \textit{b} denote entities, events, states, time, propositions and boxes, respectively. Variable names themselves are arbitrary and meaningless posing a challenge for learning. Our generator must verbalize different variable names to the same surface form. The meaning representations in Figures 1 and 2b are identical and both correspond to the same discourse except that the variables have been given different names (\textit{b}_5 in Figure 1 has been named \textit{b}_1 in Figure 2b, \textit{b}_1 is now \textit{b}_2, \textit{x}_1 is \textit{x}_2, \textit{e}_1 is \textit{e}_9, and so on).

These two problems are further compounded by the way DRSs are displayed, in a box-like format which is intuitive and easy to read but not convenient for modeling purposes. As a result, DRSs are often post-processed in a format that can be handled more easily by modern neural network models. For example, DRS variables and conditions are converted to clauses (van Noord et al., 2018) or DRSs are modified to trees where each box is a subtree and conditions within the box correspond to children of the subtree (Liu et al., 2019a, 2018).

In this paper we propose novel solutions to condition ordering and variable naming. We argue that even though DRS conditions appear unordered, they have a \textit{latent} order due to biases in the way the training data is created. To give a concrete example, the Groningen Meaning Bank (GMB; Bos et al. 2017) provides the largest collection to date of English texts annotated with DRSs. These annotations were generated with the aid of a CCG parser (Clark and Curran, 2007); atomic DRS conditions were associated with CCG supertags and then semantically combined following the syntactic CCG derivations. Even annotators creating DRSs manually would be prone to follow a canonical order (e.g., listing named entities first, then verbal predicates and their thematic roles, and finally temporal conditions). We propose a graph-based model which learns to recover the latent order of conditions without explicitly enumerating all possible orders which can be prohibitive. We also handle variable names with a method which rewrites arbitrary indices to relative ones which are in turn determined by the order of conditions.

Following previous work, we convert DRSs to a more amenable format. Specifically, we consider Discourse Representation Tree Structures (DRTSs; Liu et al. 2019b) as the semantic representation input to our document generation task, and generate a sequence of words autoregressively. We adopt an encoder-decoder framework with a treeLSTM (Tai et al., 2015) encoder and a standard LSTM (Hochreiter and Schmidhuber, 1997) decoder. Problematically, DRS trees are wide and the number of children for a given node can be as many as 180. It therefore becomes memory-consuming and sparse to assign a forget gate for each child as in the case of conventional (\textit{N}-ary) treeLSTM (Tai et al., 2015). We propose a variant which we call \textit{Sibling} treeLSTM that replaces \textit{N} forget gates with a \textit{parent} gate and a \textit{sibling} gate. As a result, it reduces...
memory usage from $O(N)$ to $O(2)$, and is more suitable for modeling wide and flat trees.

Our contributions can be summarized as follows: (1) we formalize the task of neural DRS-to-text generation; (2) we provide solutions for the problems of condition ordering and variable naming, which render generation from DRS-based meaning representations non-trivial; and (3) propose a novel sibling treeLSTM model that can be also generally used to model wide tree structures. We make our code and datasets publicly available.\footnote{https://github.com/LeonCrashCode/Discourse-Representation-Tree-Structure/tree/main/gmb/DRS-to-text}

2 Problem Formulation

Let $S$ denote a DRS-based meaning representation. The aim of DRS-to-text generation is to produce text $T$ that verbalizes input meaning $S$:

$$T^* = \arg \max_{T \in \mathcal{S}} P(T|S, \Theta),$$

where $\mathcal{S}$ is the set of all possible texts, $S$ has an arbitrary order of conditions and indexing of variables, and $\Theta$ is the set of model parameters.

Our generation model is based on the encoder-decoder framework (Bahdanau et al., 2015) and operates over tree structures. Moreover, prior to training, variable names are rewritten so that their (arbitrary) indices denote relative order of appearance. We propose a novel sibling TreeLSTM for encoding tree structures. The decoder is a sequential LSTM equipped with an attention mechanism generating word sequence $T = [t_1, t_2, ..., t_{m-1}]$, where $m$ is the length of the text. At test time, DRS conditions are normalized, i.e., they are reordered following a canonical order learned from data, and used as input to our generation model.

We first describe our DRS-to-tree conversion and variable renaming procedures (Sections 2.1 and 2.2). We next present our tree-to-sequence generation model (Section 2.3), and explain how DRS conditions are ordered (Section 2.4).

2.1 DRS-to-Tree Conversion

The algorithm of Liu et al. (2018) renders DRSs in a tree-style format. It constructs trees based on DRS conditions in the bottom box layers, without considering variables in the top layer. This results in oversimplified semantic representations and information loss (e.g., presuppositions cannot be handled). We improve upon their approach by merging variables in the top layer with variables in the bottom layer via introducing special conditions.

We collect variables in top layers of DRS boxes to construct a dictionary $d = \{v : b\}$, where $v$ denotes a variable and $b$ is a presupposition box label (e.g., $x_1 : b_1$). We then move variables from the top to the bottom layer by expressing them as special conditions $b : \text{Ref}(v)$ and placing them before conditions on variable $v$. For example, $b_5 : x_1$ in Figure 1 becomes special condition $b_5 : \text{Ref}(x_1)$ and is placed before condition $b_5 : \text{Pred}(x_1, \text{male.n.02})$ in Figure 3(a).

Once top variables have been rewritten as special conditions, the resulting DRSs are converted into trees as shown in Figure 3(b). Box variables (e.g., $b_1, b_5$) become parent nodes, while conditions, which are also subtrees, become children.
2.2 Relative Variables

We rename variables with regard to their relative position in a given DRS following a predefined traversal order.

We obtain the sequence of box variables by traversing DRSs in an outer-to-inner and left-to-right manner, e.g., \( [b_5, b_1, b_3] \) in Figure 1. For SDRSs, we replace variables in discourse relations with \( k_i \), where \( i \) denotes the \( i \)th box from left to right. For example “CONTRAST \( (b_1, b_4) \)” in Figure 1 is rewritten to “CONTRAST \( (b_0, b_1) \)”. Variables and conditions within presupposition boxes are rewritten to \( B_i \), where \( i \in \mathbb{Z} \) denotes the distance of the current box to the presupposition box.

For example, \( b_1 : \text{Agent}(e_1, x_1) \) is rewritten to \( B_0 : \text{Agent}(e_1, x_1) \) because it is in the current box \( b_1 \), while \( b_1 : \text{Pred}(x_1, \text{“male.n.02”}) \) is rewritten to \( B_{-2} : \text{Pred}(x_1, \text{“male.n.02”}) \) because it is in box \( b_3 \) and two hops away from presupposition box \( b_1 \). We use special label \( O \) for presupposition boxes pertaining to semantic content out with the current DRS. For example, \( b_0 : \text{Ref}(x_1) \) is rewritten to \( O : \text{Ref}(x_1) \) because it introduces a new presupposition box, and \( b_0 : \text{Pred}(x_1, \text{male.n.02}) \) is rewritten to \( O_0 : \text{Pred}(x_1, \text{male.n.02}) \) because the condition can only be interpreted in this new presupposition box (now \( O_0 \) and previously \( b_0 \)).

We obtain a sequence of general variables by traversing conditions as they appear in the DRS. Variables introduced for the first time are denoted by their type (going from left-to-right), while subsequent mentions of the same variables are rewritten with relative indices denoting their distance from the position where they were first introduced. Take Figure 3(a) as an example. The sequence of general variables is \( \{x_1, x_1, e_1, e_1, x_1, x_2, e_1, x_2, x_2, t_1, t_1, e_1, t_1, x_3, x_3, e_2, e_2, x_2, x_3, x_1, x_1, e_2, e_1\} \), and is rewritten to \( \{X, X_0, E, E_0, X_0, X_0, X_0, T, T_0, E_0, T_0, X, X_0, E, E_0, E_0, X_0, X_2, E_0, X_2, E_0, E_0\} \). The DRS from Figure 3(a) is shown in Figure 4 with relative variables.

2.3 Generation Model

Our generation model is based on the encoder-decoder framework, where an encoder is used to encode input DRS trees and a decoder outputs a sequence of words. A limitation of sequential encoders is that they only allow sequential information propagation without considering the structure of the input (Tai et al., 2015; Wang et al., 2019). In our case, DRS tree structures are additionally wide (the longer a document, the wider the tree) and relatively flat (see Figure 3(b)). To better model these aspects, we propose a treeLSTM encoder which takes sibling information into account.

As shown in Figure 5, the hidden representations of the sibling TreeLSTM cells are updated from preceding sibling and child nodes. More formally, the hidden representation for node \( j \) is given by:

\[
\begin{align*}
    u_j &= \tanh(g^u([x_j; h_{js}; h_{jp}])) & (1) \\
    i_j, o_j &= \sigma(g^{io}([x_j; h_{js}; h_{jp}])) & (2) \\
    f_{js} &= \sigma(g^{f_j}([x_j; h_{js}])) & (3) \\
    f_{jp} &= \sigma(g^{f_p}([x_j; h_{jp}])) & (4) \\
    c_j &= i_j \cdot u_j + f_{js} \cdot c_{js} + f_{jp} \cdot c_{jp} & (5) \\
    h_j &= o_j \cdot \tanh(c_j), & (6)
\end{align*}
\]

where \( x_j \) is the token input representation, \( h_{js} \) is the hidden representation of the sibling node preceding \( j \), \( h_{jp} \) is the hidden representation of the last child of node \( j \) (Equation (1)), \( g^u \) are linear functions, and \( \sigma \) is a sigmoid function (Equations (2)–(4)). For each node \( j \), we obtain its cell input representation \( u_j \) (Equation (1)), its input gate \( i_j \) and output gate \( o_j \) (Equation (2)), and two forget gates \( f_{js} \) (Equation (3)) and \( f_{jp} \) (Equation (4)) for its neighbor cell and the last child cell, respectively. The memory of the current cell \( c_j \) (Equation (5)) is updated by the gated sum of its cell input representation and the memories of its neighbor and child cells. The hidden representation of current node \( h_j \) is computed with its output gate \( o_j \) (Equation (6)).
Finally, a DRS is represented by the hidden representations of its nodes \([h_0, h_1, ..., h_n] \) as computed by the sibling treeLSTM \( (n\prime)\) denotes the number of nodes). The decoder is a standard LSTM with global attention (Bahdanau et al., 2015).

### 2.4 Condition Ordering

As discussed previously, DRSs at test time may exhibit an arbitrary order of conditions, which our model should be able to handle. Our solution is to reorder conditions prior to generation by learning a latent canonical order from training data (e.g., to recover boxes \( b_1 \) and \( b_2 \) in Figure 1 from boxes \( b_1 \) and \( b_3 \) in Figure 2). More formally, given a set of conditions \( R_{set} \), we obtain an optimal ordering \( R = [r_0, r_1, ..., r_{n-1}] \) such that:

\[
R^* = \arg \max_{R \in \pi(R_{set})} \text{SCORE}_{\mathbb{K}}(R|R_{set}), \tag{7}
\]

where \( \pi(R_{set}) \) are all permutations of \( R_{set} \), and \( R^* \) is the order with the highest likelihood according to \( \text{SCORE}_{\mathbb{K}} \). Here, \( \mathbb{K} \) parametrizes \( \text{SCORE} \) as “knowledge” we collect from our training data by observing canonical orders of conditions. Unfortunately, the time complexity of calculating Equation (7) is \( O(n!) \), we must enumerate all possible permutations for a set of conditions with \( n \) as large as 180. Since this is prohibitive, we resort to graph ordering which allows us to recover the order of the conditions without enumeration.

**Graph Construction** We construct a graph from the set of DRS conditions which we break down into graph nodes and edges. Conditions in DRSs can be simple or complex according to their type of arguments. A simple condition might have a relation name with two arguments (e.g., Named \( (x_3, \text{“tom”}) \) and Agent \( (e_1, x_3) \)), while a complex condition has a scoped name (e.g., possibility \( \phi \)) and takes one or more DRSs as arguments. Simple conditions are denoted by a 3-tuple \( (l_s, a_0, a_1) \), where \( l_s \) is the condition name (e.g., Named and Agent) and \( a_0 \) and \( a_1 \) are its first and the second argument, respectively, which could be a variable or constant (e.g., \( e_1, x_3 \) and “piano.n.01”). Complex conditions are a 2-tuple \( (l_c, V_r) \), where \( l_c \) is the scope name, and \( V_r \) the set of arguments scoped by the condition. For example, the set of arguments for the possibility scope \( \phi \) in Figure 1 is \( \{e_1, e_2, x_1, x_3, \text{“tom”}, \text{“stop.v.05”}, \text{“male.n.”}\} \).

Condition names become nodes in our graph. Simple conditions are further divided into constant and thematic nodes. Constant nodes are constructed by concatenating the relation name in the condition with the constant argument (e.g., condition \( \text{Pred}(x_1, \text{“male.n.02”}) \) becomes node \( \text{Pred “male.n.02”} \)). Thematic nodes correspond to the relation name of the thematic condition (e.g., Agent \( (e_1, x_1) \) becomes the node “Agent”). Complex nodes correspond to the name of complex conditions (e.g., possibility \( \phi \)).

We insert edges between graph nodes if these share arguments. For example, in Figure 6(b), there is an edge connecting node \( \text{Pred “male.n.02”} \) with Agent as they share argument \( x_1 \). We label this edge with \( a_1 \) to denote the fact that it is the second argument of Agent. Another edge is drawn between \( \text{Pred “play.v.03”} \) and Agent (as they share argument \( e_1 \) with label \( a_0 \) denoting that this is the first argument of Agent. Edges between nodes are bidirectional, with inverse edges bearing the suffix “-of”. Edges drawn between constant and complex nodes bear the label “Related”, while edges between two constant nodes (with the same variables) bear the label “Equal” (we provide a more formal description in the Appendix).

**Ordering Model** Given graph \( G = (R_{set}, E) \), where \( R_{set} = \{ r_0, r_1, ..., r_{n-1} \} \) is the set of nodes and \( E \) is the set of edges in \( G \), our model outputs \( R^* \) as the optimal order of \( R_{set} \).

As shown Figure 6(a), each node is a sequence of words. A BiLSTM is applied to obtain representation \( x_i \) of each node \( r_i = \{w_0^i, ..., w_{m_i}^i\} \):

\[
x_i = \text{BiLSTM}(\{w_0^i, ..., w_{m_i}^i\}). \tag{8}
\]

We encode the graph with a Graph Convolutional Recurrent network (GCRN; Seo et al. 2018). For
each node $r_i$, we collect information from neighbor hidden representations with a gate controlling the information flow from neighbors to current nodes:

$\mathbf{h}_k^i = \sum_j g_j^k \cdot \mathbf{h}_{ij}^{k-1};$ \hspace{1cm} (9)

$g_j^k = \sigma(f(e_{ji}, \mathbf{h}_i^{k-1}, \mathbf{h}_{ij}^{k-1})),$ \hspace{1cm} (10)

where $e_{ji}$ is the embedding of edges from node $r_j$ to $r_i$, and $k$ is the recurrent step in the GRU. The node hidden representations are updated as:

$\mathbf{h}_i^k = \text{GRUCell}([\mathbf{x}_i; g_{Gj}^{k-1}], \mathbf{h}_i^{k-1})$ \hspace{1cm} (11)

$g_j^k = \text{GRUCell}(\frac{1}{n_i} \sum_i h_i^k, g_G^{k-1})$ \hspace{1cm} (12)

where $g_{Gj}$ represents the hidden representation of the graph as the average of (hidden) node representations, and GRUCell denotes the gated recurrent cell function. We obtain the hidden representations of nodes in the final recurrent step $(K)$ as $H^K = \{ h_0^K, h_1^K, ..., h_{n-1}^{K-1} \}$.

Our decoder obtains the orders with the highest probability. We avoid enumerating all possible permutations for a set of nodes by generating their order autoregressively with an LSTM-based Pointer Network (PN; Vinyals et al. 2015):

\[
\text{SCORE}_G(R|R_{set}) = \text{PN}(R|R_{set}, H^K, \theta) \hspace{1cm} (13)
\]

\[
\text{PN}(R|R_{set}, H^K, \theta) = \prod_i \text{P}(r_i|r_{<i}, H^K) \hspace{1cm} (14)
\]

\[
\text{P}(r_i|r_{<i}, H^K) = \text{softmax}(v^T \tanh(W[h_i^d; H^K])) \hspace{1cm} (15)
\]

where $\theta$ are the parameters of the Pointer Network, $h_i^d$ is the $i$th step hidden representation of the Pointer Network, and $v$, and $W$ are parameters. Hidden representation $h_i^d$ is updated by the input representation of the $(i-1)$th ordered node: $h_i^d = \text{LSTMCell}(x_{r_{i-1}}, h_{r_{i-1}}^d)$. All parameters are optimized with standard back-propagation.

3 Experiments

Our experiments were carried out on the Groningen Meaning Bank (GMB; Bos et al. 2017) which provides a large collection of English documents annotated with DRSs. We used the standard training, development, and test splits that come with the distribution of the corpus. All DRSs in the GMB were preprocessed into the tree-based format discussed in Section 2.1. We also extracted from the training data conditions and their order for training our graph ordering model. Dataset statistics are shown in Table 1.

| Task                  | train | dev  | test  |
|-----------------------|-------|------|-------|
| Generation            | 7,970 | 992  | 1,038 |
| Condition Ordering    | 133,332 | 16,493 | 17,624 |

Table 1: GMB dataset statistics; number of documents (generation) and number of different sequences of conditions (ordering).

3.1 Condition Ordering

Models and Settings Before evaluating our generator per se, we assess the effectiveness of the proposed condition ordering model (see Section 2.4). Specifically we compare four kinds of graphs: NoEdges, is a graph without edges; FullEdges, is a complete graph where each pair of nodes has edges; SiGraph, is the proposed graph without bidirectional edges; and BiGraph, is the proposed graph with bidirectional edges (see Figure 6). We also consider Counting, a baseline model which greedily orders pairs of conditions according to their frequency of appearance in the training data (see the Appendix for details).

For all neural models the embedding dimension was 50 and the hidden dimension 300. The bidirectional LSTM used for representing the graph nodes has a single layer, and the recurrent step in the GCRN is 2 ($K = 2$). We applied the Adam optimizer (Kingma and Ba, 2014). We use accuracy to measure the percentage of absolute orders which are predicted correctly and Kendall’s $\tau$ coefficient to measure the relationship between two lists of ordered items; $\tau$ ranges from $-1$ to 1, where $-1$ means perfect inversion and 1 means perfect agreement.

Results Table 2 summarizes our results. SiGraph performs better than NoEdges (+14.83% accuracy), showing that edge information is helpful for the representation of nodes which are used to order conditions. FullEdges performs worse than SiGraph (−13.68% accuracy), underlying the fact that graph structure matters (i.e., edges are helpful when connecting certain pairs of nodes). BiGraph achieves the best ordering performance by a large margin compared to SiGraph (+9.63 % accuracy). One possible reason is that bidirectionality ensures all nodes have incoming edges, which can be used to update the node representations.
### 3.2 Ideal-World Generation

**Models and Settings** We first examine generation performance in an ideal setting where (gold standard) condition orders are given and the indices of variables are fixed.

We compared the proposed treeLSTM against **Seq**, a baseline sequence-to-sequence model which adopts a bidirectional LSTM as its encoder. Trees were linearized in a top-down and left-to-right fashion. \( X = [x_0, x_1, ..., x_{n-1}] \), where \( n \) is the tree length. We obtained hidden representations \( H = [h_0, h_1, ..., h_{n-1}] \) of the input with:

\[
[h_0, h_1, ..., h_{n-1}] = \text{BiLSTM}([x_0, x_1, ..., x_n])
\]

In addition, we included various models with tree-based encoders: **ChildSum**, is the bidirectional chidsum-treeLSTM encoder of Tai et al. (2015); it operates over right-branch binarized trees; **Nary**, is the bidirectional Nary-TreeLSTM of Tai et al. (2015), again over right-branch binarized trees; and **Sibling** is our bidirectional sibling-TreeLSTM. All models were equipped with the same LSTM decoder, global attention (Bahdanau et al., 2015), and the copy strategy of See et al. (2017).

The embedding dimension was 300 and the hidden dimension 512. All encoders and decoders have 2 layers. The detailed settings are shown in the Appendix. We measure generation quality with case-insensitive BLEU (Papineni et al., 2002).

#### Results

Table 3 shows our results on the development dataset. Overall, treeLSTM models perform better (average +1.69 BLEU) than sequence models. Nary performs better (+0.26 BLEU) than ChildSum because the latter cannot model the order of children. Sibling performs best (74.22 BLEU), because it not only encodes the tree structure but also keeps track of sequential information.

### 3.3 Real-World Generation

**Models and Settings** We finally, present our results in a more realistic setting where both problems of condition ordering and variable naming must be addressed. We recover condition order using four approaches: a **Naive** method which has no special-purpose ordering mechanism; the order of conditions is random in the development/test sets and fixed in the training set; **Random**, the order of conditions is random in the training, development, and test sets; **Counting**, the order of conditions is recovered by the Counting method; **GraphOrder** recovers the order of conditions with BiGraph. All comparison systems employ variable renaming as introduced in Section 2.2. We report experiments with a sequence-to-sequence generator and our sibling-TreeLSTM.

#### Results

Table 4 shows our results on the development dataset. For Random, we report average results after shuffling 5 times (variance shown in parentheses).

Table 5: Real-world generation (test set).
Results Table 4 summarizes our results on the development set. Naive performs poorly, indicating that both Seq and Sibling models are sensitive to the order of conditions. Random, has higher variance with Seq (+16.51) compared to Sibling. Hidden representations for each timestep in Seq are heavily influenced by all previous steps, which are sequentially encoded; subtrees are encoded as a unit in Sibling, which is a more global representation for capturing patterns. Overall, we observe that the order of conditions plays a key role in the generation: both Seq and Sibling models improve when ordering of conditions is explicitly incorporated (either with Counting or GraphOrder). We observe that the combination of Sibling with GraphOrder achieves the best results (58.73 BLEU).

Table 5 presents our results on the test set. We compare our Sibling encoder against a sequential one. Both models are interfaced with GraphOrder. We also compare to a previous graph-to-text model (Song et al., 2018; Damonte and Cohen, 2019) which has been used for generating from AMRs. We converted DRSs to graphs following the method of Liu et al. (2020); graphs were encoded with a GCRN (Seo et al., 2018) and decoded with an LSTM. As can be seen, Sibling+GraphOrder outperforms all comparison systems achieving a BLEU of 59.26. However, compared to ideal-world generation (see Table 3) there is still considerable room for improvement.

3.4 Analysis
Figure 7 shows model performance on test set against DRS size (i.e., the number of nodes in a DRS tree). Perhaps unsurprisingly, we see that generation quality deteriorates with bigger DRSs (i.e., with >1,600 nodes).

While BLEU is frequently adopted as an automatic evaluation metric for generation tasks, it is somewhat problematic in our case as it merely calculates word overlap between generated and gold-standard text without assessing whether model output is faithful to the semantics of the input (i.e., the DRS meaning representations). To this effect, we present examples of text generated by our model, demonstrating how the DRS input constrains and affects the output text.

Figure 8 shows examples of text generation from the test set. In the first example, the model generates the word because from the rhetorical relation, 

$$\text{BECAUSE}(b_{10}, b_{12})$$

Temporal information (highlighted in blue in the figure) is also accurately reflected in the generated text (sell is inflected to its present tense form). In addition, the model tends to over-generate (e.g., the word dollar is mentioned twice) and sometimes misses out on important determiners (e.g., some). In the second example, the model generates the word themselves referring to the entities mentioned before, e.g., $x_{29}$ equals to $x_{27}$ which refers to inmates, resolving the coreference. In the third example, the model generates all for food and goods corresponding to the Implication (IMP) condition (i.e., $\forall x (P(x) \rightarrow Q(x))$).

4 Related Work
Much previous work has focused on text generation from formal representations of meaning focusing exclusively on isolated sentences or queries. The literature offers a collection of approaches to generating from AMRs most of which employ neural models and structured encoders (Song et al., 2018; Beck et al., 2018; Damonte and Cohen, 2019; Ribeiro et al., 2019; Zhu et al., 2019; Cai and Lam, 2020; Wang et al., 2020). Other work generates text from structured query language (SQL) adopting either sequence-to-sequence (Iyer et al., 2016) or graph-to-sequence models (Xu et al., 2018).

Basile (2015) was the first to attempt generation from DRT-based meaning representations. He proposes a pipeline system which operates over graphs and consists of three components: an alignment module learns the correspondence between surface text and DRS structure, an ordering module determines the relative position of words and phrases in the surface form and a realizer generates the final text. Narayan and Gardent (2014) simplify complex sentences with a two-stage model which
first performs sentence splitting and deletion operations over DRSs and then uses a phrase-based machine translation model for surface realization.

Our work is closest to Basile (2015); we share the same goal of generating from DRSs, however, our model is trained end-to-end and can perform long-form generation for documents and sentences alike. We also adopt an ordering component, but we order DRS conditions rather than lexical items, and propose a model capable of inferring a global order. There has been long-standing interest in information ordering within NLP (Lapata, 2003; Abend et al., 2015; Chen et al., 2016; Gong et al., 2016; Logeswaran et al., 2018; Cui et al., 2018; Yin et al., 2019; Honovich et al., 2020). Our innovation lies in conceptualizing ordering as a graph scoring task which can be further realized with graph neural network models (Wu et al., 2020).

5 Conclusions

In this paper, we have focused on document-level generation from formal meaning representations. We have adopted DRT as our formalism of choice and highlighted various challenges associated with the generation task. We have introduced a novel sibling treeLSTM for encoding DRSs rendered as trees and shown it is particularly suited to trees with wide branches. We have experimentally demonstrated that our encoder coupled with a graph-based condition ordering model outperforms strong comparison systems. In the future, we would like to embed our generator in practical applications such as summarization and question answering.

Acknowledgments

We thank the anonymous reviewers for their feedback. We gratefully acknowledge the support of the European Research Council (Lapata, Liu; award number 681760), the EU H2020 project SUMMA (Cohen, Liu; grant agreement 688139) and Bloomberg (Cohen, Liu).

References

Omri Abend, Shay B. Cohen, and Mark Steedman. 2015. Lexical event ordering with an edge factored model. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Tech-
nologies (NAACL-HLT), pages 1161–1171, Denver, Colorado, USA.

Nicholas Asher and Alex Lascarides. 2003. *Logics of conversation*. Cambridge University Press.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Diego, California.

Valerio Basile. 2015. *From Logic to Language: Natural Language Generation from Logical Forms*. Ph.D. thesis, University of Groningen, Netherlands.

Valerio Basile and Johan Bos. 2011. Towards generating text from discourse representation structures. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 145–150, Nancy, France.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 273–283, Melbourne, Australia.

Johan Bos, Valerio Basile, Kilian Evang, Noortje J Vincent, and Johannes Bjerva. 2017. The Groningen meaning bank. In *Handbook of Linguistic Annotation*, pages 463–496. Springer.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, pages 7464–7471, New York, USA.

John Carroll and Stephan Oepen. 2005. High-efficiency realization for a wide-coverage unification grammar. In Robert Dale and Kam-Fai Wong, editors, *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP)*, volume 3651, pages 165–176. Springer, Jeju, Korea.

Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. 2016. Neural sentence ordering. *arXiv preprint arXiv:1607.06952*.

Stephen Clark and James Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. 2018. Deep attentive sentence ordering network. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4340–4349, Brussels, Belgium.

Marco Damonte and Shay B. Cohen. 2019. Structural neural encoders for AMR-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 3649–3658, Minneapolis, USA.

Federico Fancellu, Sorcha Gilroy, Adam Lopez, and Mirella Lapata. 2019. Semantic graph parsing with recurrent neural network DAG grammars. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2769–2778, Hong Kong, China.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Database*. MIT Press, Cambridge, MA.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime G. Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 731–739, San Diego, California, USA.

Jingjing Gong, Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. 2016. End-to-end neural sentence ordering using pointer network. *arXiv preprint arXiv:1611.04953*.

Valerie Hajdik, Jan Buys, Michael Wayne Goodman, and Emily M. Bender. 2019. Neural text generation from rich semantic representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2259–2266, Minneapolis, Minnesota.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Or Honovich, Lucas Torroba Hennigen, Omri Abend, and Shay B. Cohen. 2020. Machine reading of historical events. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7486–7497, Online. Association for Computational Linguistics.

Matic Horvat, Ann Copestake, and Bill Byrne. 2015. Hierarchical statistical semantic realization for Minimal Recursion Semantics. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS)*, pages 107–117, London, UK. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2073–2083, Berlin, Germany.
Hans Kamp. 1981. A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof, editors, *Formal Methods in the Study of Language*, volume 1, pages 277–322. Mathematisch Centrum, Amsterdam.

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic: An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht.

Diederik P. Kingma and Jimmy Ba. 2014. *Adam: A method for stochastic optimization*. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, Banff, Canada.

Ioannis Konstas, Srinivasa Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. *Neural AMR: Sequence-to-sequence models for parsing and generation*. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 146–157, Vancouver, Canada.

Mirella Lapata. 2003. *Probabilistic text structuring: Experiments with sentence ordering*. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, pages 545–552, Sapporo, Japan.

Jiangming Liu, Shay B. Cohen, and Mirella Lapata. 2018. *Discourse representation structure parsing*. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 429–439, Melbourne, Australia.

Jiangming Liu, Shay B. Cohen, and Mirella Lapata. 2019a. *Discourse representation parsing for sentences and documents*. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6248–6262, Florence, Italy.

Jiangming Liu, Shay B. Cohen, and Mirella Lapata. 2019b. *Discourse representation structure parsing with recurrent neural networks and the transformer model*. In *Proceedings of the IWCS Shared Task on Semantic Parsing*, Gothenburg, Sweden. Association for Computational Linguistics.

Jiangming Liu, Shay B. Cohen, and Mirella Lapata. 2020. *Discorer: A fast evaluation metric for discourse representation structure parsing*. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4547–4554.

Lajanugen Logeswaran, Honglak Lee, and Dragomir Radev. 2018. *Sentence ordering and coherence modeling using recurrent neural networks*. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 5285–5292, New Orleans, Louisiana, USA.

Wei Lu and Hwee Tou Ng. 2011. A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1611–1622, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Shashi Narayan and Claire Gardent. 2014. *Hybrid simplification using deep semantics and machine translation*. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 435–445, Baltimore, Maryland. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. *Bleu: a method for automatic evaluation of machine translation*. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Leonardo F.R. Ribeiro, Claire Gardent, and Iryna Gurevych. 2019. *Enhancing AMR-to-text generation with dual graph representations*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3174–3185, Hong Kong, China.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. *Get to the point: Summarization with pointer-generator networks*. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1073–1083, Vancouver, Canada.

Youngjoo Seo, Michäel Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. *Structured sequence modeling with graph convolutional recurrent networks*. In *Neural Information Processing Systems (NeurIPS)*, pages 362–373, Montréal, Canada. Springer.

Stuart M. Shieber, Gertjan van Noord, Fernando C. N. Pereira, and Robert C. Moore. 1990. *Semantic-head-driven generation*. *Computational Linguistics*, 16(1):30–42.

Linfeng Song, Yue Zhang, Zhiqiu Wang, and Daniel Gildea. 2018. *A graph-to-sequence model for AMR-to-text generation*. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1616–1626, Melbourne, Australia.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. *Improved semantic representations from tree-structured long short-term memory networks*. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 1556–1566, Beijing, China.
Sho Takase, Jun Suzuki, Naoaki Okazaki, Tsutomu Hi-rao, and Masaaki Nagata. 2016. Neural headline generation on Abstract Meaning Representation. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1054–1059, Austin, Texas. Association for Computational Linguistics.

Rob A. Van der Sandt. 1992. Presupposition projection as anaphora resolution. Journal of Semantics, 9(4):333–377.

Rik van Noord, Lasha Abzianidze, Antonio Toral, and Johan Bos. 2018. Exploring neural methods for parsing discourse representation structures. Transactions of the Association for Computational Linguistics, 6:619–633.

Rik van Noord, Antonio Toral, and Johan Bos. 2019. Linguistic information in neural semantic parsing with multiple encoders. In Proceedings of the 13th International Conference on Computational Semantics (IWCS), pages 24–31, Gothenburg, Sweden.

Noortje J. Venhuizen, Johan Bos, Petra Hendriks, and Harm Brouwer. 2018. Discourse semantics with information structure. Journal of Semantics, 35(1):127–169.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In Neural Information Processing Systems (NeurIPS), pages 2692–2700, Montréal, Canada.

Juen-tin Wang. 1980. On computational sentence generation from logical form. In Proceedings of the 8th International Conference on Computational Linguistics (COLING), pages 405–411, Tokyo, Japan.

Tianming Wang, Xiaojun Wan, and Hanqi Jin. 2020. AMR-to-text generation with graph transformer. Transactions of the Association for Computational Linguistics, 8:19–33.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 1060–1070, Hong Kong, China.

Michael White, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with CCG. In Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT), pages 267–276, Copenhagen, Denmark.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems.
A Counting Method

We count how frequently condition \( r_i \) appears before \( r_j \) with type \( t \). Type \( t \) is identified according to the overlap between the arguments of the two conditions\(^6\). For example, \( r_i = (\text{Name}, x_3, \text{“tom”}) \) and \( r_j = (\text{Agent}, e_1, x_3) \) has the type “\( a_0 \rightarrow a_1 \)”, showing that the first argument in \( r_i \) equals to the second argument in \( r_j \). We score the order of two conditions using the following function:

\[
\text{SCORE}_K(R|\mathcal{R}_{\text{set}}) = \sum_i (\sum_{i<j} \text{COUNT}_K(r_i, r_j)) - \sum_{j<i} \text{COUNT}_K(r_i, r_j)
\]  

(16)

where \( \text{COUNT} \) returns the frequency of a pair of conditions subject to a dataset or corpus \( K \); the score increases with \( r_j \) following \( r_i \) more frequently than preceding it.

A.1 Types

We define different types of relations between two conditions based on argument overlap (i.e., two simple conditions, a simple and a complex condition, and two complex conditions).

Simple and Simple  Given two simple conditions, \( r_i = (l_i, a_{0i}, a_{1i}) \) and \( r_j = (l_j, a_{0j}, a_{1j}) \) (with different arguments), we define their types as:

- \( t = a_0 \rightarrow a_0 \) if \( a_{0i} = a_{0j} \) and \( a_{1i} \neq a_{1j} \).
- \( t = a_1 \rightarrow a_1 \) if \( a_{0i} \neq a_{0j} \) and \( a_{1i} = a_{1j} \).
- \( t = a_0 \rightarrow a_0, a_1 \rightarrow a_1 \) if \( a_{0i} = a_{0j} \) and \( a_{1i} = a_{1j} \).
- \( t = a_0 \rightarrow a_1 \) if \( a_{0i} = a_{1j} \) and \( a_{1i} \neq a_{0j} \).
- \( t = a_1 \rightarrow a_0 \) if \( a_{1i} = a_{0j} \) and \( a_{0i} \neq a_{1j} \).
- \( t = a_0 \rightarrow a_1, a_1 \rightarrow a_0 \) if \( a_{1i} = a_{0j} \) and \( a_{0i} = a_{1j} \).
- \( t = \text{None} \) if others

Simple and Complex  Given a simple conditions \( r_i = (l_i, a_{0i}, a_{1i}) \) and a complex condition \( r_j = (l_j, V_j) \), the types are defined as:

- \( t = 1 \) if \( a_{0i} \in V_j \) and \( a_{1i} \notin V_j \).
- \( t = 1 \) if \( a_{0i} \notin V_j \) and \( a_{1i} \in V_j \).
- \( t = 2 \) if \( a_{0i} \in V_j \) and \( a_{1i} \in V_j \).
- \( t = \text{None} \) if others

Complex and Complex  Given two complex conditions, \( r_i = (l_i, V_i) \) and \( r_j = (l_j, V_j) \), the types are defined as:

- \( t = \text{intersection} \) if \( V_i \cap V_j \neq \emptyset \).
- \( t = \text{None} \) otherwise

A.2 Greedy Algorithm

We generate an ordering of conditions following greedy Algorithm 1 which identifies the highest scoring pair in \( \mathcal{R}_{\text{set}} \), appends it to partial ordering \( R^* \), and keeps going until \( \mathcal{R}_{\text{set}} \) is empty. The score is given by the function:

\[
\text{PARTIAL}_K(r) = \sum_{r' \in \mathcal{R}_{\text{set}}} \text{COUNT}(r, r')_K - \sum_{r'' \in R} \text{COUNT}(r, r'')_K,
\]  

(17)

B Edge Construction Algorithm

Algorithm 2 shows how edges are created for graph ordering.

C Model Settings

In the following, we report the best experimental settings for our condition ordering and text generation models.

C.1 Condition Ordering Models

Model hyperparameters are shown in Table 6. The size of the token embeddings (in the graph nodes) and edge embeddings is 100. The node hidden dimension is the same as the hidden dimension of the BiLSTM which is used to encode a sequence of input words for each node and the hidden dimension of PointerNet. The BiLSTM and PointerNet have one layer. Training hyperparameters are shown in Table 7.
Algorithm 2 Edges Construction

Input: $N_t$ thematic nodes; $N_c$, complex nodes; $N_e$, constant nodes.
Output: $E$, set of edges

1: $E = []$
2: for $n_t, n_e$ in $N_t, N_e$ do
3: if $n_t.a_0 = n_e.a_0$ then
4: $E = E \cup \{n_t \xrightarrow{a_0} n_e\}$
5: $E = E \cup \{n_e \xrightarrow{a_0,a_t} n_t\}$
6: end if
7: if $n_t.a_1$ is in $n_e.V$ then
8: $E = E \cup \{n_e \xrightarrow{Related} n_t\}$
9: $E = E \cup \{n_t \xrightarrow{Related} n_e\}$
10: end if
11: end for
12: for $n_c, n_e$ in $N_c, N_e$ do
13: if $n_c.a_0$ in $n_e.V$ then
14: $E = E \cup \{n_e \xrightarrow{Related} n_c\}$
15: $E = E \cup \{n_c \xrightarrow{Related} n_e\}$
16: end if
17: end for
18: for $n_e, n'_e$ in $N_e, N_e$ do
19: if $n_e.a_0$ is $n'_e.a_0$ and $n_e$ not is $n'_e$ then
20: $E = E \cup \{n_e \xrightarrow{Equal} n'_e\}$
21: end if
22: end for

| hyperparameters   | value |
|-------------------|-------|
| embedding dim     | 50    |
| hidden dim        | 300   |
| LSTM layer        | 1     |
| graph step        | 2     |
| beam size         | 64    |

Table 6: Hyperparameters of condition ordering model.

C.2 Text Generation Models

Model hyperparameters are shown in Table 8. The size of the input embeddings in the encoder and the decoder is 300. The hidden dimensions of the encoder and decoder are 512. Both the encoder and decoder have two layers. The hyperparameters of the training are shown in Table 9.

D Hyperparameter Tuning

We show below model performance with various hyperparameters. Best hyperparameters were manually chosen after monitoring model accuracy on the development set.

We take BiGraph as our final condition ordering model (see Table 10) and Sibling with BiGraph as the DRS-to-text generation model (see Table 11).

| hyperparameters   | value             |
|-------------------|-------------------|
| optimizer         | adam              |
| learning rate     | 0.001             |
| $\beta$           | (0.9, 0.999)      |
| dropout rate      | 0.5               |
| batch size        | 32                |
| max gradient norm | 5                 |
| training steps    | 400,000           |
| learning rate decay| 0.3              |
| start decay steps | 200,000           |
| decay steps       | 20,000            |
| warmup steps      | 40,000            |

Table 7: Training hyperparameters for the condition ordering models.

| hyperparameters   | value |
|-------------------|-------|
| embedding dim     | 300   |
| hidden dim        | 512   |
| layer             | 2     |
| attention         | general |
| beam size         | 5     |

Table 8: Hyperparameters of text generation model.

| hyperparameters   | value          |
|-------------------|----------------|
| optimizer         | adam           |
| learning rate     | 0.001          |
| $\beta$           | (0.9, 0.999)   |
| dropout rate      | 0.5            |
| batch size        | 30000          |
| batch type        | tokens         |
| max gradient norm | 5              |
| training steps    | 30,000         |
| learning rate decay| 0.5            |
| start decay steps | 8,000          |
| decay steps       | 1,000          |
| warmup steps      | 4,000          |

Table 9: Training hyperparameters for text generation model.
| embedding | hidden | step | dropout | beam | Acc  |
|-----------|--------|------|---------|------|------|
| 50        | 300    | 2    | 0.5     | 64   | 69.10|
| 100       | —      | —    | —       | —    | 68.61|
| 150       | —      | —    | —       | —    | 67.76|
| —         | 100    | —    | —       | —    | 67.33|
| —         | 200    | —    | —       | —    | 68.77|
| —         | —      | 1    | —       | —    | 55.71|
| —         | —      | 3    | —       | —    | 69.12|
| —         | —      | —    | 0.2     | —    | 68.02|
| —         | —      | —    | 0.4     | —    | 68.52|
| —         | —      | —    | 0.6     | —    | 67.83|
| —         | —      | —    | —       | 16   | 69.03|
| —         | —      | —    | —       | 32   | 69.07|
| —         | —      | —    | —       | 96   | 69.10|

Table 10: Performance of condition ordering model (development set; various hyperparameters). Increasing model size leads to out-of-memory problems when training on a single GPU.

E Examples

We provide example output of our final model (Sibling+GraphOrder) on the GMB test dataset. The DRS in tree format with condition ordering given by GraphOrder is shown in Figures 9–11. Figure 10 expands nonterminal $b_{33}$ in Figure 9, and Figure 11 expands nonterminal $b_{28}$ in Figure 10. The corresponding document generated by Sibling is:

the u.s. dollar hit a record low against the euro tuesday. it took a dollar, but 48 cents to buy one euro, and a series of problems including the key of the u.s. housing sector in which has been battered by the slowing n.01, including continuing the u.s. economy, the dollar. speculator are selling dollars because they expect that the u.s. central bank will try to stimulate the economy by cutting interest rates soon. u.s. lower interest rates can cut the return on investments. the falling dollar is prompting oil-rich nations around the persian gulf to consider ending the practice of linking the value of its currency to those of the dollar and instead supplement the u.s. currency. such a move would reduce demand for dollars and weaken the u.s. currency.
Table 11: Performance of text generation model (development set; various hyperparameters). Increasing model size leads to out-of-memory problems when training on a single GPU.
Figure 9: A partial DRS in tree format with condition ordering recovered by GraphOrder.
Figure 11: A partial DRS in tree format with condition ordering recovered by GraphOrder.