Learning to Map Context-Dependent Sentences to Executable Formal Queries (Supplementary Material)

Alane Suhr[†], **Srinivasan Iyer**[‡], **and Yoav Artzi**[†]

[†] Dept. of Computer Science and Cornell Tech, Cornell University, New York, NY
{suhr, yoav}@cs.cornell.edu
[‡] Paul G. Allen School of Computer Science & Engineering, Univ. of Washington, Seattle, WA
sviyer@cs.washington.edu

A Data Processing

Date and Number Resolution We replace instances of spelled-out multi-digit numbers in the original data (e.g., *flight two three five*) with a numerical representation (e.g., *flight 235*). We resolve expressions containing references to dates using UWTime (Lee et al., 2014). Interactions in ATIS are annotated with the date they took place, which we use as document time. We use the *newswire* annotator in UWTime to annotate each utterance in ATIS with date tags, and add one week to any predicted dates that occur before the document date. This heuristic follows the assumption that users always ask for information in the future. UWTime is able to predict dates contained in the gold queries in the training data with approximately 70% accuracy. Without using interaction dates and resolving date expressions, the model would not be able to generate correct dates, unless through overfitting to the training set. Previous work on ATIS addresses the problem of resolving relative date expressions by modifying the output queries. For example, Zettlemoyer and Collins (2009) add logical constants such as *tomorrow* to the lambda-calculus lexicon. To the best of our knowledge, no previous context-dependent work on ATIS uses interaction dates to recover the referents of date expressions.

Entity Anonymization We replace known entities in user utterances and SQL queries with anonymized tokens. Using the database, we generate a set of known entities and their natural language and SQL forms, for example the set of city names from the `city` table. When anonymizing an example, we first identify entities that occur in the input sequence, and replace each with a unique anonymized token, using the same token for entities that occur multiple times. We also identify numerical constants, which include flight numbers and times, as entities in the input sequence. This process gives us a set of entities in the input sequence and their corresponding anonymized tokens. During training, we replace any tokens in the gold SQL query that are in this set with the appropriate anonymized token. Entity anonymization is separate of entity scoring, described in Section 6, which computes scores for generating anonymized tokens independently of generating raw SQL tokens. When entity scoring is ablated in our experiments, entity anonymization is still applied as a pre-processing technique, but anonymized tokens with indices are used as regular members of the input and output vocabularies.

Post-processing After generating a query but before evaluating it against the SQL database or comparing with the gold labels, we post-process it. We de-anonymize all generated anonymized tokens using the anonymization set extracted from the input sequences, and correct mismatched parentheses by adding closing parenthesis at the end of the query.

B Copying Segments

Segment Extraction from Previous Queries To construct $S(\bar{y})$, we deterministically extract subtrees of the SQL parse tree from $\bar{y}$, as well as `SELECT` statements containing special modifiers like `MIN`. We use the `sqlparse` package to construct a tree, and recursively traverse its subtrees. We consider all subtrees of a `WHERE` clause. We separate children of conjunctions into distinct subtrees. When evaluating with predicted queries, we extract segments from the most recent prediction that has correct syntax and follows the database structure.

Alignment of Segments with Gold Queries During training, we align the set of extracted segments $S(\bar{y})$ with the gold query to construct a new

query that contains references to extracted segments. We first extract known entities, e.g. city names, from the current utterance $\bar{x}_i$, using the entity set described above. We greedily substitute the longest extracted segments in $S(\bar{y})$ first. If segment $\bar{s}$ is a subsequence of the gold query $\bar{y}_i$, we replace that subsequence with a reference to it. We do not replace the subsequence if it contains one of the entities extracted from the input sequence; entities mentioned in the current utterance should be explicitly generated in the current prediction, as these are most likely not references to previous constraints.

C Implementation Details

Learning We use the ADAM optimizer (Kingma and Ba, 2014) with an initial global learning rate of 0.001. The batch size $B = 16$. We use patience as a stopping mechanism, with an initial patience of 10 epochs. We compute loss and token-level accuracy on a held-out validation set after each epoch. This set includes 5% of the training data, and when using our split of the dataset, does not contain scenarios that are present in the remaining 95% of the data. We use the same validation set across all of our experiments. After an epoch, if token-level loss on the validation set increases since the previous epoch, the global learning rate is multiplied by 0.8. When token-level accuracy on the validation set increases to a maximum value during training, we multiply the current patience by 1.01. During training, we apply dropout with probability 0.5 after the first decoder LSTM layer, and after computing $\mathbf{m}_k$ at each decoding step. The model parameters used to evaluate on the development and test sets are those that yielded the highest string-level validation accuracy.

During training, when multiple gold labels are present, we use the shortest. Loss is not computed for utterance-query pairs if, after pre-processing and query segment alignment, the gold label contains more than 200 tokens. However, if using the turn-level encoder, this pair's input sequence is encoded and the turn-level state is updated. During evaluation on the development and test sets, we limit generation to 300 tokens.

If not using the turn-level encoder, we delimit the previous and current utterances with a special delimiter token when encoding the inputs (Section 4.2). The corresponding encoder hidden states of the delimiters are not used during attention.

Split	Model	Strict Denotation
Original	FULL	$68.4_{\pm 0.2}$
	– preprocess.	$67.1_{\pm 1.0}$ **(-1.3)**
Ours	FULL	$62.5_{\pm 0.9}$
	– preprocess.	$53.0_{\pm 8.5}$ **(-9.5)**

Table 4: Strict table accuracy results using our model with and without pre-processing on the development sets of the original data split and our data split.

Model	Query	Denotation	
		Relaxed	Strict
FULL-GOLD	$42.1_{\pm 0.8}$	$66.6_{\pm 0.7}$	$66.1_{\pm 0.7}$
– turn-level encoder	$42.5_{\pm 1.7}$	$66.3_{\pm 1.7}$	$65.7_{\pm 1.9}$
– batch re-weighting	$41.1_{\pm 0.7}$	$65.5_{\pm 0.3}$	$64.8_{\pm 0.6}$
– input pos. embs.	$38.0_{\pm 0.4}$	$61.4_{\pm 1.1}$	$60.5_{\pm 1.1}$
– anon. scoring	$40.8_{\pm 0.7}$	$64.7_{\pm 1.4}$	$63.9_{\pm 1.3}$
– pre-processing	$35.3_{\pm 6.9}$	$57.9_{\pm 8.4}$	$57.4_{\pm 8.2}$

Table 5: Ablations on FULL-GOLD, showing performance on the development set averaged over all utterances. Gold queries are provided for previous query segment extraction. In each model, $h = 3$. We show average and standard deviation over five trials for each model.

Parameters We use word embeddings of size 400. The word embeddings are not pre-trained. Utterance age embeddings are of size 50. Query segment age embeddings are of size 64. For all models that use query segment copying, $g = 4$. All LSTMs have a hidden size of 800. The sizes of the learned matrices are: $\mathbf{W}^{\mathbf{A}} \in \mathbb{R}^{850\times800}$, $\mathbf{W}^{m} \in \mathbb{R}^{1650\times800}$, $\mathbf{W}^{o} \in \mathbb{R}^{800\times|V^o|}$, $\mathbf{b}_w^o \in \mathbb{R}^{|V^o|}$, and $\mathbf{W}^{S} \in \mathbb{R}^{800\times1600}$. Unless otherwise noted, the initial hidden state and cell memory of all LSTMs are zero-vectors. All parameters are initialized randomly from a uniform distribution $U[-0.1, 0.1]$.

D Results

D.1 Overfitting in Original Data Split

We assess overfitting on the training set of the original split of the data by measuring how performance changes when data pre-processing is removed. Table 4 shows that removing data pre-processing lowers the performance of FULL by around 9% when using our data split. However, on the original split of the data, performance drops by only 1.3%. This relatively high performance is only possible due to learned biases within the scenarios.

D.2 Ablations using Gold Previous Queries

Table 5 shows results on ablating components from FULL when extracting segments from previous gold queries.

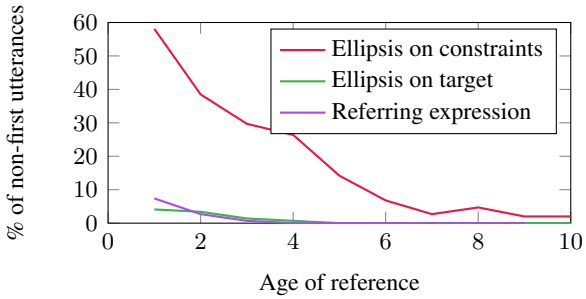

Figure 6: Distribution of referent ages over sampled utterances in the development set.

E Analysis

E.1 Data Analysis

The two most common contextual phenomena in ATIS are ellipsis and referring expressions. Ellipsis refers to utterances that omit relevant information mentioned in the past. For example, $\bar{y}_2$ in Figure 2 contains flight endpoint constraints that are not present in $\bar{x}_2$. To recover these constraints, the model must resolve an implicit reference to $\bar{x}_1$. Referring expressions explicitly refer to earlier constraints or system responses. In the example utterances, the phrase *which ones* in $\bar{x}_3$ refers to the table returned by executing $\bar{y}_2$.

An analysis of twenty development-set interactions shows that utterances after the first turn contain on average 1.85 omitted constraints, and 60.1% of utterances after the first turn omit at least one constraint. Figure 6 shows the distribution of referent ages over all utterances in the twenty analyzed interactions from the development set, considering three major types of references: ellipsis on constraints, ellipsis on the attributes targeted by the user request (e.g., if a user omits the target attribute in the current utterance, but it is clear from context), and referring expressions (explicit mentions of previous constraints). 9% of utterances after the first utterance omit the target attribute; 11% contain a referring expression to previous results or constraints.

E.2 Attention Analysis

Figures 7, 8, and 9 demonstrate the robustness of the full approach when attending over multiple previous utterances. Each figure shows attention while processing the same example, which is the third utterance in an interaction from the development set. This interaction exemplifies a user focus state change. To process the third utterance, *show flights*, correctly, the model must be able to recover constraints mentioned in the first utterance, although the user briefly changes focus in the second utterance. In each figure, the query is shown on the lefthand side, and the utterances are at the bottom. The opacity of each cell represents the attention probability for each token during each generation step. Darker lines in the figure separate the three utterances.

Figure 7 shows the attention computed by FULL when provided with gold query segments. The decoder attends over entities in the previous utterance, including the flight endpoints and date, when generating the query. Figure 8 shows the attention computed by FULL-0 when provided with gold query segments. This model does not have explicit access to the constraints mentioned in the first utterance. It is unable to recover these constraints, and instead makes up constraints, such as endpoints and a date, while also making a semantic mistake, `city.city_name = 1200`. This demonstrates the robustness provided by the attention mechanism in the full model. For comparison, Figure 9 shows the attention computed by S2S+ANON on the same example. Like FULL, this model attends over entities in the first utterance, including flight endpoints and the date. Both FULL and S2S+ANON were able to recover the correct query.

Figures 10, 11, and 12 demonstrate how the ability to copy query segments is critical to our model's performance. FULL and FULL-0 are able to recover the correct query. In both cases, `SEGMENT_9`, which is extracted from the previous query, contains the flight endpoint constraints (from the first utterance), as well as a constraint that the flight be the shortest one available (from the second utterance). S2S+ANON is unable to recover the minimum-time constraint, even though it has the ability to attend over the relevant tokens in the second utterance. The ability of FULL-0 to recover this constraint without attending on previous utterances demonstrates the benefit that copying previous segments provides. These examples also show that when copying query segments, fewer decoding steps are required.

E.3 Contextual Analysis

We construct several example interactions targeting the contextual phenomena discussed in Section 9, and test FULL against them.

$\bar{x}_1$: show me flights from seattle to denver after 6am $\bar{x}_2$: leaving after 7am $\bar{x}_3$: leaving after 8am $\bar{x}_4$: leaving before 9am $\bar{x}_5$: leaving after 10am $\bar{x}_6$: leaving from san francisco

This example shows ellipsis of both constraints (flight endpoints) and target attribute (flights) while modifying existing constraints. FULL is able to predict correct queries for all new utterances as the user continues to change constraints and elided values increase in age. Whether $\bar{y}_6$ includes a time constraint or not is ambiguous; FULL generates a query to search for all flights from San Francisco to Denver regardless of time.

$\bar{x}_1$: show me flights from seattle to denver $\bar{x}_2$: leaving after 7am $\bar{x}_3$: stopping in san francisco $\bar{x}_4$: on american airlines $\bar{x}_5$: which is the cheapest $\bar{x}_6$: with breakfast

This example shows ellipsis of both constraints and target attribute while adding constraints. FULL is able to predict correct queries for all utterances in this interaction.

$\bar{x}_1$: show me flights from seattle to denver after 6am $\bar{x}_2$: how much does it cost $\bar{x}_3$: what meal is offered $\bar{x}_4$: which airlines are available $\bar{x}_5$: what type of airplane does it use $\bar{x}_6$: what ground transportation is available

This example shows ellipsis on constraints (endpoints and time) while changing the target attribute. Additionally, it demonstrates change in focus while the user switches from asking for flights to asking about airlines. While ambiguous, the final utterance $\bar{x}_6$ is interpreted as finding ground transportation in Denver. FULL is able to recover correct queries for all utterances in this interaction.

$\bar{x}_1$: show me flights from seattle to denver after 6am $\bar{x}_2$: what ground transportation is available in denver $\bar{x}_3$: i want to fly on delta

This example shows ellipsis when user focus changes, temporarily rendering the origin city and time constraints irrelevant. FULL predicts queries for the first two utterances correctly, but fails to generate the correct origin city constraint in the third prediction. Instead, it generates a constraint that Denver is the origin city. When $\bar{x}_2$ is removed from the interaction, both predictions are correct.

References

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Kenton Lee, Yoav Artzi, Jesse Dodge, and Luke Zettlemoyer. 2014. Context-dependent semantic parsing for time expressions. In *Proceedings of the Association for Computational Linguistics*. https://doi.org/10.3115/v1/P14-1135.

Luke S. Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the AFNLP*. http://www.aclweb.org/anthology/P09-1110.

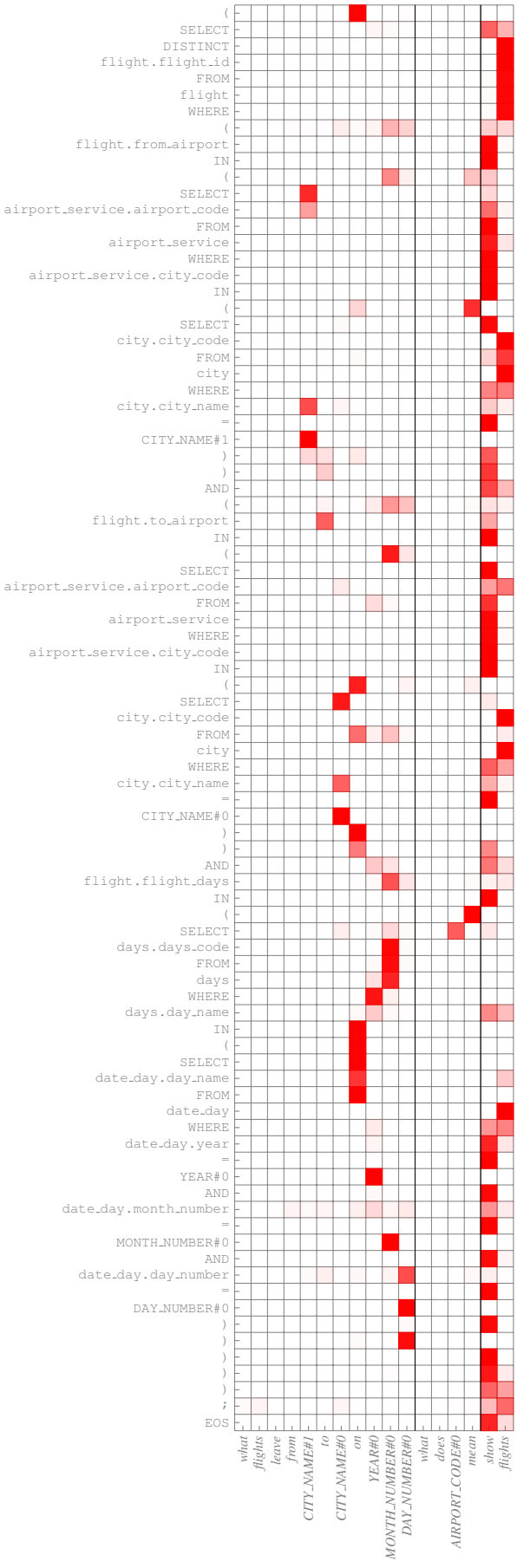

Figure 7: Attention computed by FULL after a user state focus change. Compare to Figures 8 and 9.

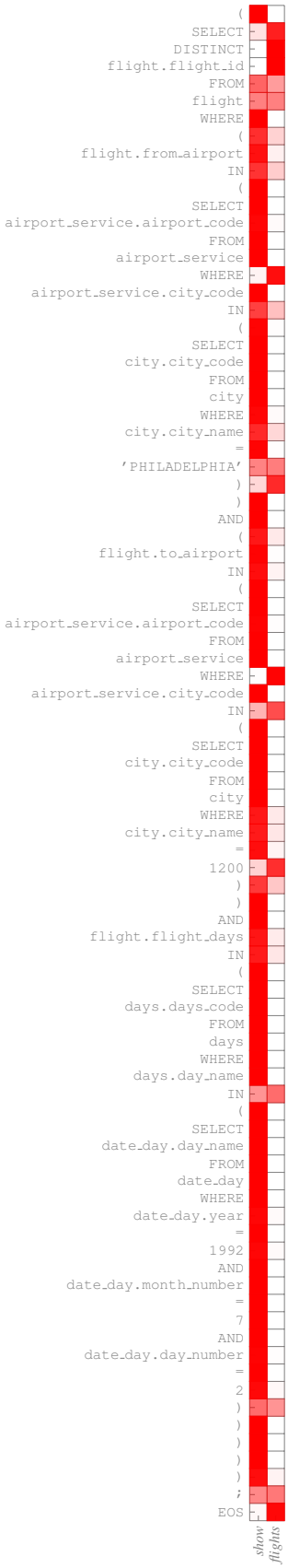

Figure 8: Attention computed by FULL-0 when generating a query for an utterance after a user state focus change. The model does not have explicit access to flight endpoint or date constraints, and is unable to generate the correct query. Figure 7 shows FULL on this example; Figure 9 shows S2S-ANON.

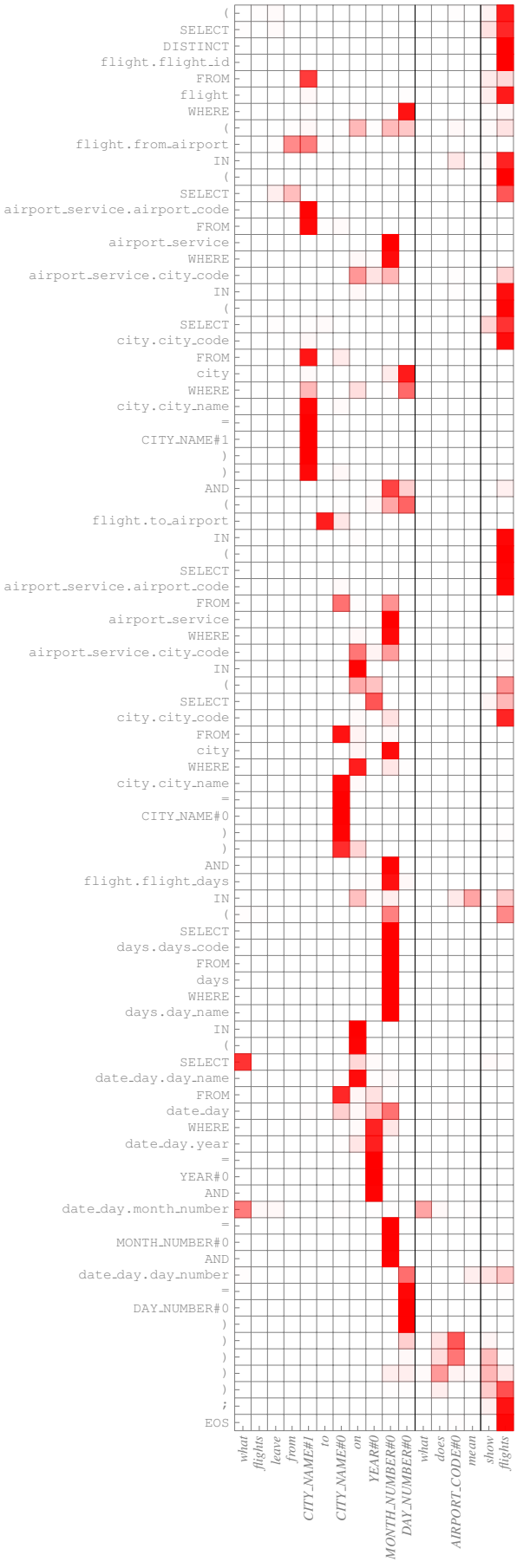

Figure 9: Attention computed by S2S+ANON after a user state focus change. Compare to Figures 7 and 8.

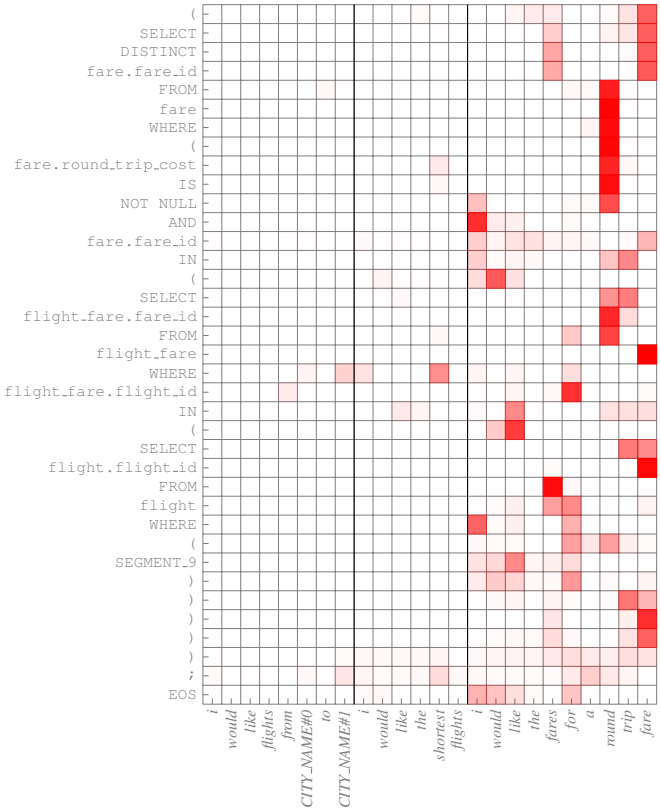

Figure 10: Attention computed by FULL, demonstrating copying of query segments. Figure 11 shows FULL-0 on this example; Figure 12 shows S2S-ANON.

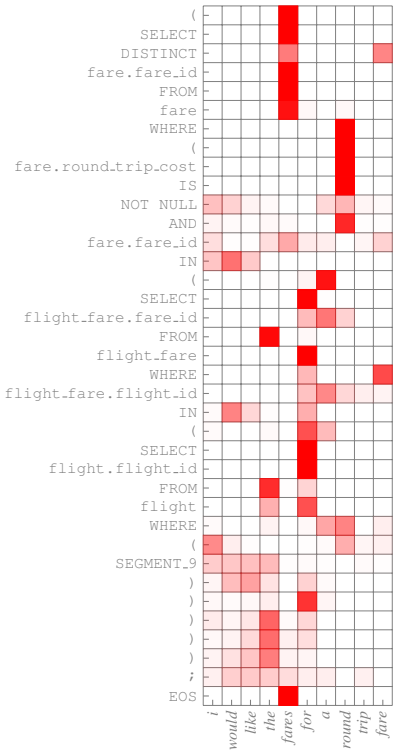

Figure 11: Attention computed by FULL-0 that makes use of query segments to recover constraints it does not otherwise have explicit access to. Figure 10 shows FULL on this example; Figure 12 shows S2S-ANON.

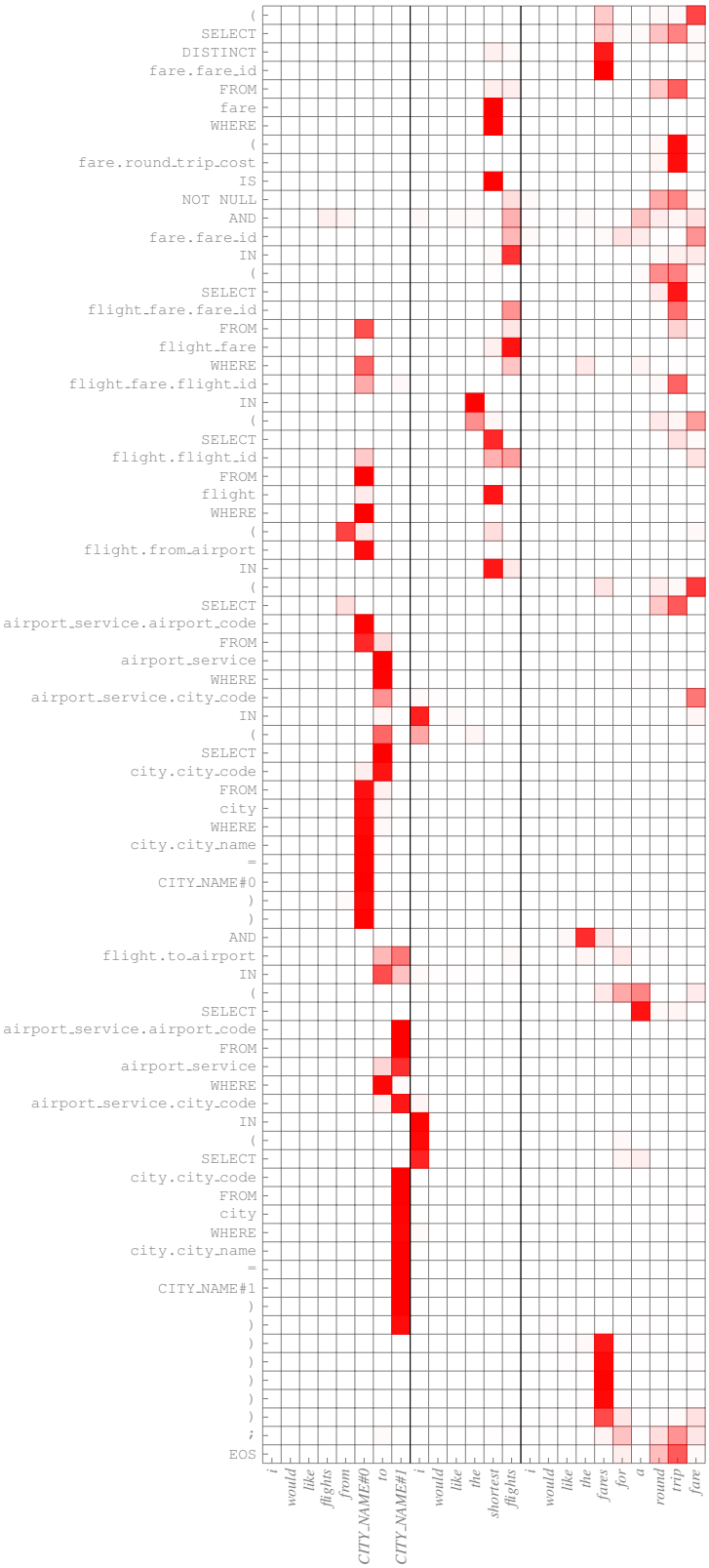

Figure 12: Attention computed by S2S-ANON, demonstrating a failure in recovering constraints. Figure 10 shows FULL on this example; Figure 11 shows FULL-0.