A Sample Grammar and Data Generation Grammar

A sample SCFG is given in Table 5 to facilitate the definition of $domain$.

This sample SCFG is not to be confused with the grammar used to generate our dataset. In comparison, the SCFG for data generation contains a larger number of entities, relations, and operations than the sample SCFG. Selected rules from the person domain and restaurant domain are provided below in Table 7 and Table 8.

B Contents of the Memory

In this paper, we assume the memory is provided to the model by a human expert. We picked the strategy of selecting one example for each grammar rule g in the synchronous context free grammar G that generated the data. It makes the formulation of maximum conditional likelihood learning straightforward, because this strategy produces a memory such that for any given utterance there is a unique sequence of *look-up* and *adapt* actions that produce the correct logical form. Recall that an example of a grammar rule g is an *utterance-logical form* pair generated using g and other rules of G. In particular, we choose an example for g such that the top level predicate of the example's logical form matches the top level predicate of the target of g. For example, we will choose to put

⟨ John 's parents, `(field parent john)` ⟩

in memory for the grammar rule

$$g_1 := \textbf{FIELD} \rightarrow \langle \textbf{PSN}_2 \text{ 's } \textbf{PSN\_REL}_1, \texttt{(field } \textbf{PSN\_REL}_1 \textbf{ PSN}_2\texttt{)} \rangle$$

because `field` is the root predicate of both the example logical form `(field parent john)` and the target of the rule `(field` $\textbf{PSN\_REL}_1$ $\textbf{PSN}_2$`)`. Here's another example. For a terminal rule like

$$g_2 := \textbf{PSN} \rightarrow \langle John, \texttt{(person john)} \rangle$$

we put the pair ⟨ John, `(person john)` ⟩ in memory, because the top level predicate of the logical form (`person`) matches that of g_2's target (`person`). We would not put for instance ⟨ John 's children, `(field children john)`⟩ because the top level predicate of the logical form (`field`) does not match that of g_2's target (`person`).

We do note that the requirement of manually selected examples to put in memory is a big limitation of the current approach. In future work, we wish to automate the building of memory.

C The trace of an example run of Look-up and Adapt

A trace of an example run of the LOOKUPANDADAPT algorithm is given in Table 9. The utterance of this run is based on the sample SCFG grammar given in Table 5. Given a memory with 3 examples and a new utterance "John's Parents", the LOOKUPANDADAPT algorithm will look-up an example from the memory that matches the attended parts of the new utterance and recursively adapt the logical form of the retrieved example to produce a correct logical form that fits the new utterance.

D Statistics of the generated data

We provide a breakdown of the number of examples used in training and evaluation for the six domains of discourse at different depths in Table 6. The definition for the different groups of examples and their use is explained in § 5 for dataset generation.

Left-hand-side		Source(Utterance)	Target(Logical Form)
PSN	→	John	`john`
PSN	→	Mary	`mary`
PSN_REL	→	parents	`parent`
PSN_REL	→	children	`child`
FIELD	→	**PSN_REL**$_1$ of **PSN**$_2$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
FIELD	→	**PSN**$_2$'s **PSN_REL**$_1$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
S	→	**PSN**$_1$	**PSN**$_1$
S	→	**PSN_REL**$_1$	**PSN_REL**$_1$
S	→	**FIELD**$_1$	**FIELD**$_1$

Table 5: Sample SCFG G_{sample}. The subscripts indicate linking of terminals/nonterminals in source and target. A SCFG rule can only be applied to linked terminals/nonterminals together.

		domain of discourse					
		person	restaurant	event	course	animal	vehicle
D_{old}	full	737	810	911	960	801	729
	d=2	417	490	591	640	481	409
	d=3	320	320	320	320	320	320
E_{olddev}	full	96	96	96	96	96	96
	d=2	65	68	67	66	61	64
	d=3	31	28	29	30	35	32
$E_{oldtest}$	full	96	96	96	96	96	96
	d=2	63	60	61	62	67	64
	d=3	33	36	35	34	29	32
E_{newdev}	full	96	96	96	96	96	96
	d=2	68	67	58	68	60	66
	d=3	28	29	38	28	36	30
$E_{newtest}$	full	96	96	96	96	96	96
	d=2	60	61	70	60	68	62
	d=3	36	35	26	36	28	34
M_{old}	full	23	16	16	16	16	15
	d=1	20	13	13	13	13	12
	d=2	2	2	2	2	2	2
	d=3	1	1	1	1	1	1
M_{new}	full	27	20	20	20	20	19
	d=1	24	17	17	17	17	16
	d=2	2	2	2	2	2	2
	d=3	1	1	1	1	1	1

Table 6: Dataset Statistics: sizes of each component with depth breakdown. Please refer to § 5 for the precise definition of each row of the table. In the *extension* scenario, for each domain of discourse d, the model is trained on examples from $D_{old,d}$ using $M_{old,d}$ as memory and is evaluated on examples from $E_{newdev,d}$ and $E_{newtest,d}$ using $M_{new,d}$ as memory. In the *transfer* scenario, for each domain of discourse d, the model is trained on $\cup_{d' \neq d} D_{old,d'}$ using $\cup_{d' \neq d} M_{old,d'}$ as memory, and is evaluated on $E_{olddev,d}$ and $E_{oldtest,d}$ using $\cup_{\forall d'} M_{old,d'}$ as memory.

Left-hand-side		Source(Utterance)	Target(Logical Form)
PSN	→	john	`(person john)`
PSN	→	mary	`(person mary)`
PSN	→	colleagues	`(person colleague)`
PSN	→	professors	`(person professor)`
PSN_REL	→	parents	`(relation parent)`
PSN_REL	→	children	`(relation child)`
PSN_REL	→	hometown	`(relation hometown)`
PSN_REL	→	salary	`(relation salary)`
FIELD	→	**PSN_REL**$_1$ of **PSN**$_2$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
FIELD	→	**PSN**$_2$'s **PSN_REL**$_1$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
CMD	→	set **FIELD**$_1$ to **VALUE**$_2$	`(set` **FIELD**$_1$ **VALUE**$_2$`)`
VALUE	→	**FIELD**$_1$	**FIELD**$_1$
VALUE	→	**PSN**$_1$	**PSN**$_1$
S	→	**FIELD**$_1$	**FIELD**$_1$
S	→	**CMD**$_1$	**CMD**$_1$

Table 7: This is a selected subset of the grammar used to generate data for the *person* domain. Rules not shown here include those for introducing flexibility in language and rules for additional entities and relations.

Left-hand-side		Source(Utterance)	Target(Logical Form)
RST	→	chinese restaurant	`(restaurant chinese)`
RST	→	italian restaurant	`(restaurant italian)`
RST	→	french restaurant	`(restaurant french)`
RST	→	german restaurant	`(restaurant german)`
RST_REL	→	address	`(relation address)`
RST_REL	→	reviews	`(relation reviews)`
RST_REL	→	dishes	`(relation dishes)`
RST_REL	→	price	`(relation price)`
FIELD	→	**RST_REL**$_1$ of **RST**$_2$	`(field` **RST_REL**$_1$ **RST**$_2$`)`
FIELD	→	**PSN**$_2$'s **RST_REL**$_1$	`(field` **RST_REL**$_1$ **RST**$_2$`)`
CMD	→	set **FIELD**$_1$ to **VALUE**$_2$	`(set` **FIELD**$_1$ **VALUE**$_2$`)`
VALUE	→	**FIELD**$_1$	**FIELD**$_1$
VALUE	→	**RST**$_1$	**RST**$_1$
S	→	**FIELD**$_1$	**FIELD**$_1$
S	→	**CMD**$_1$	**CMD**$_1$

Table 8: This is a selected subset of the grammar used to generate data for the *restaurant* domain. Rules not shown here include those for introducing flexibility in language and rules for additional entities and relations.

Func	Description	New utterance and current logical form	Memory
LKUPNADPT	Initial call always has uniform attention.	**[John 's parents]** None	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
LOOKUP	Retrieves entry $\star$ from memory by utterance similarity.	**[John 's parents]** `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ $\star$ ⟨Mary 's parents , `(field parent mary)`⟩
::ADAPT	Adapt first child. `parent` is aligned to **parents**. YES, they match, keep `parent`. There's no children. Return.	John 's **[parents]** `(field [parent] mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
::ADAPT	Adapt second child. `mary` is aligned to **John**. NO, they don't match. Find replacement and return it.	**[John]** 's parents `(field parent [mary])`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
::::LKUPNADPT	Find replacement for `mary`.	**[John]** 's parents `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
::::LOOKUP	Retrieves entry $\star$ from memory by utterance similarity.	**[John]** 's parents `(field parent mary)`	$\star$ ⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
::::LKUPNADPT	There's no children. Return `john`	**[John]** 's parents `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩
LKUPNADPT	Replace `mary` with `john`. Exhausted all children, return the current logical form.	**[John 's parents]** `(field parent john)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary 's parents , `(field parent mary)`⟩

Table 9: Trace of a sample run of the algorithm with sample memory, utterance, and attention. The attention weights (argument $\boldsymbol{w}'$ to LOOKUPANDADAPT and variable $\boldsymbol{w}$ in ADAPT) are visualized as brackets. Bracketed words in the *utterance* receive more attention, un-bracketed words receive less attention. For ADAPT calls, bracketed words in the *logical form* indicate the current subtree of the whole logical form tree that the algorithm is looking at (the argument `T` to ADAPT). Colons are used to indicate the depth of the call stack.