Conversational Semantic Parsing for Dialog State Tracking

Jianpeng Cheng **Devang Agrawal** **Héctor Martínez Alonso** **Shruti Bhargava** **Joris Driesen**
Federico Flego **Dain Kaplan** **Dimitri Kartsaklis** **Lin Li** **Dhivya Piraviperumal**
Jason D Williams **Hong Yu** **Diarmuid Ó Séaghdha** **Anders Johannsen**
Apple
{jianpeng.cheng,hmartinezalonso,shruti_bhargava,joris_driesen,
fflego,dain_kaplan,dkartsaklis,lli9,dhivyaprp,jason_williams4,
hong_yu,doseaghdha,ajohannsen}@apple.com

Abstract

We consider a new perspective on dialog state tracking (DST), the task of estimating a user's goal through the course of a dialog. By formulating DST as a semantic parsing task over hierarchical representations, we can incorporate semantic compositionality, cross-domain knowledge sharing and co-reference. We present TreeDST, a dataset of 27k conversations annotated with tree-structured dialog states and system acts.[1] We describe an encoder-decoder framework for DST with hierarchical representations, which leads to 20% improvement over state-of-the-art DST approaches that operate on a flat meaning space of slot-value pairs.

1 Introduction

Task-based dialog systems, for example digital personal assistants, provide a linguistic user interface for all kinds of applications: from searching a database, booking a hotel, checking the weather to sending a text message. In order to understand the user, the system must be able to both parse the meaning of an utterance and relate it to the context of the conversation so far. While lacking the richness of a conversation between two humans, the dynamics of human-machine interaction can still be complex: the user may change their mind, correct a misunderstanding or refer back to previously-mentioned information.

Language understanding for task-based dialog is often termed "dialog state tracking" (DST) (Williams et al., 2016), the mental model being that the intent of the user is a partially-observed state that must be re-estimated at every turn given new information. The dialog state is typically modelled as a set of independent slots, and a standard DST system will maintain a distribution over values for each slot. In contrast, language understanding for other NLP applications is often formulated as semantic parsing, which is the task of converting a single-turn utterance to a graph-structured meaning representation. Such meaning representations include logical forms, database queries and other programming languages.

These two perspectives on language understanding—DST and semantic parsing—have complementary strengths and weaknesses. DST targets a fuller range of conversational dynamics but typically uses a simple and limiting meaning representation. Semantic parsing embraces a *compositional* view of meaning. By basing meaning on a space of combinable, reusable parts, compositionality can make the NLU problem space more tractable (repeated concepts must only be learned once) and more expressive (it becomes possible to represent nested intents). At the same time, most semantic parsing research treats a sentence as an isolated observation, detached from conversational context.

This work unifies the two perspectives by reformulating DST as *conversational semantic parsing*. As in DST, the task is to track a user's goal as it accumulates over the course of a conversation. The goal is represented using a structured formalism like those used in semantic parsing. Specifically, we adopt a hierarchical representation which captures domains, verbs, operators and slots within a rooted graph grounded to an ontology. The structured dialog state is capable of tracking nested intents and representing compositions in a single graph (Turn 5 Table 1). The formalism also naturally supports cross-domain slot sharing and cross-turn co-reference through incorporating the shared slots or the references as sub-graphs in the representation (Turn 3 Table 1).

Using a reverse annotation approach inspired

[1]The dataset is available at https://github.com/apple/ml-tree-dst.

Turn	Utterance and Annotation
1	*Hi can you book me a flight to Paris please.*
	`user.flight.book.object.equals`
	`  .destination.equals.location.equals.Paris`
	Sure, when and where will you depart?
	`system.prompt.flight.book.object.equals`
	`  .source`
	`  .departureDateTime`
2	*Tomorrow from London*
	`user.flight.book.object.equals`
	`  .destination.equals.location.equals.Paris`
	`  .source.equals.location.equals.London`
	`  .departureDateTime.equals`
	`    .date.equals.definedValue.equals.Tomorrow`
	I found 5 flights for you. The earliest one departs at 10 AM with a cost of £105. Would you like it?
	`system`
	`  .prompt.flight.book.object.equals`
	`    .departureDateTime.equals.time`
	`  .inform.flight.find`
	`    .count.equals.5`
	`    .object.equals`
	`      .departureDateTime.equals.time.equals`
	`        .hour.equals.10`
	`        .meridiem.equals.AM`
	`      .price.equals.105`
3	*Do I have any calendar event on that day?*
	`user.calendarEvent.checkExistence.object.equals`
	`  .dateTimeRange.equals`
	`    .date.equals.definedValue.equals.Tomorrow`
	No you don't have any event.
	`system.inform.calendarEvent.find.notExisted`
4	*I would book the 10 AM flight for me please*
	`user.flight.book.object.equals`
	`  .destination.equals.location.equals.Paris`
	`  .source.equals.location.equals.London`
	`  .departureDateTime.equals`
	`    .date.equals.definedValue.equals.Tomorrow`
	`    .time.equals`
	`      .hour.equals.10`
	`      .meridiem.equals.AM`
	Here is your booking information. Please confirm.
	`system.offer.flight.book`
5	*Direction to my next meeting.*
	`user.navigation.find.object.equals`
	`  .destination.equals.reference`
	`    .calendarEvent.object.equals`
	`      .listOffset.equals.1`
	Here is the direction to your meeting at the GlassHouse.
	`system`
	`  .navigation.inform.find`
	`  .calendarEvent.inform.find.object.equals`
	`    .location.equals.GlassHouse`

Table 1: An example conversation in TreeDST with annotations. User and system utterances are marked in red and blue respectively. We use *dot* to represent tree edges and increased indentation levels to reveal multiple children attached to the same parent node. A side-by-side comparison between a dotted tree and its full drawing can be found in Appendix A.

by Shah et al. (2018) and Rastogi et al. (2019), we have collected a large dataset of task-oriented dialogs annotated with hierarchical meaning representations. Each dialog was generated through a two-step process. First, a generative dialog simulator produces a meaningful conversational flow and a template-based utterance for each turn in the conversation. Then the utterances are paraphrased by human annotators to render more realistic and natural conversations. The resulting dataset, which we call TreeDST, covers 27k conversations across 10 domains. Conversations in TreeDST are non-linear: they contain glitches which represent system failures and uncooperative user behaviors such as under- and over-specifying slot information. There are also use cases not addressed in existing slot-filling datasets, including compositional intents and multi-intent utterances.

The second contribution of this work is a conversational semantic parser that tackles DST as a graph generation problem. At each turn, the model encodes the current user utterance and representations of dialog history, based upon which the decoder generates the updated dialog state with a mixture of generation and copy mechanism. In order to track practical conversations with intent switching and resumption (Lee and Stent, 2016; El Asri et al., 2017), we adopt a stack (Rudnicky and Xu, 1999) to represent multiple tasks in the dialog history, and a parent-pointer decoder to speed up decoding. We conducted controlled experiments to factor out the impact of hierarchical representations from model architectures. Overall our approach leads to 20% improvement over state-of-the-art DST approaches that operate on a flat meaning space.

2 Related Work

Modeling Traditional DST models apply discriminative classifiers over the space of slot-value combinations (Crook and Lemon, 2010; Henderson et al., 2014b; Williams et al., 2016). These models require feature extraction from user utterances based on manually constructed semantic dictionaries, making them vulnerable to language variations. Neural classification models (Mrkšić et al., 2017; Mrkšić and Vulić, 2018) alleviate the problem by learning distributed representations of user utterances. However, they still lack scalability to large unbounded output space (Xu and Hu, 2018; Lee et al., 2019) and structured representations. To address the limitations, some recent work treats slot filling as a sequence generation task (Ren et al., 2019; Wu et al., 2019).

On the other hand, single-turn semantic parsers have long used structured meaning representations to address compositionality (Liang et al., 2013; Banarescu et al., 2013; Kollar et al., 2018; Yu et al., 2018; Gupta et al., 2018). Solutions range from chart-based constituency parsers (Berant et al., 2013) to more recent neural sequence-to-sequence models (Jia and Liang, 2016). The general challenge of scaling semantic parsing to DST is that dialog state, as an accumulation of conversation history, requires expensive context-dependent annotation. It is also unclear how utterance semantics can be aggregated and maintained in a structured way. In this work we provide a solution to unify DST with semantic parsing.

Data Collection The most straightforward approach to building datasets for task-oriented dialog is to directly annotate human-system conversations (Williams et al., 2016). A limitation is that the approach requires a working system at hand, which causes a classic chicken-and-egg problem for improving user experience. The issue can be avoided with Wizard-of-Oz (WoZ) experiments to collect human-human conversations (El Asri et al., 2017; Budzianowski et al., 2018; Peskov et al., 2019; Byrne et al., 2019; Radlinski et al., 2019). However, dialog state annotation remains challenging and costly in WoZ, and the resulting distribution could be different from that of human-machine conversations (Budzianowski et al., 2018). One approach that avoids direct meaning annotation is to use a dialog simulator (Schatzmann et al., 2007; Li et al., 2016). Recently, Shah et al. (2018) and Rastogi et al. (2019) generate synthetic conversations which are subsequently paraphrased by crowdsourcing. This approach has been proven to provide a better coverage while reducing the error and cost of dialog state annotation (Rastogi et al., 2019). We adopt a similar approach in our work, but focusing on a structured meaning space.

3 Setup

3.1 Problem Statement

We use the following notions throughout the paper. A conversation X has representation Y grounded to an ontology $\mathcal{K}$: at turn t, every user utterance x_t^u is annotated with a *user dialog state* y_t^u, which represents an accumulated user goal up to the time step t. Meanwhile, every system utterance x_t^s is annotated with a *system dialog act* y_t^s, which represents the system action in response to y_t^u. Both y_t^u and y_t^s adopt the same structured semantic formalism to encourage knowledge sharing between the user and the system. From the perspective of the system, y_t^s is observed (the system knows what it has just said) and y_t^u must be inferred from the user's utterance. For the continuation of an existing goal, the old dialog state will keep being updated; however, when the user proposes a completely new goal during the conversation, a new dialog state will overwrite the old one. To track goal switching and resumption, a stack is used to store non-accumulable dialog states in the entire conversation history (in both data simulation and dialog state tracking).

There are two missions of this work: 1) building a conversational dataset with structured annotations that can effectively represent the joint distribution $P(X, Y)$; and 2) building a dialog state tracker which estimates the conditional distribution of every dialog state given the current user input and dialog history $P(y_t^u | x_t^u, X_{<t}, Y_{<t})$.

3.2 Representation

We adopt a hierarchical and recursive semantic representation for user dialog states and system dialog acts. Every meaning is rooted at either a `user` or `system` node to distinguish between the two classes. Non-terminals of the representation include `domains`, user `verbs`, system `actions`, `slots`, and `operators`. A `domain` is a group of activities such as creation and deletion of calendar events. A user `verb` represents the predicate of the user intent sharable across domains, such as `create`, `delete`, `update`, and `find`. A system `action` represents a type of system dialog act in response to a user intent. For example, the system could `prompt` for a slot value; `inform` the user about the information they asked for; and `confirm` if an intent or slot is interpreted correctly. Nested properties (e.g., time range) are represented as a hierarchy of `slot-operator-argument` triples, where the `argument` can be either a sub-slot, a terminal `value` node or a special `reference` node. The `value` node accepts either a categorical label (e.g., day of week) or an open value (e.g., content of text message). The `reference` node allows a whole intent to be attached as a slot value, enabling the construction of cross-domain use cases (e.g., Turn 5 of Table 1). Meanwhile, co-reference to single slots is directly achieved by subtree copying (e.g., Turn 3 of Table 1). Finally, conjunction is implicitly supported by allowing a set of arguments to be attached to the same slot. Overall, the representation presented above focuses on ungrounded utterance semantics to decouple understanding from execution. By incorporating domain-specific logic, the representation can be mapped to executable programs at a later stage.

4 Data Elicitation

4.1 Overview

We sidestep the need for collecting real human-system conversations and annotating them with complex semantics by adopting a reverse data elicitation approach (Shah et al., 2018; Rastogi et al., 2019). We model the generative process $P(X,Y) = P(Y)P(X|Y)$, where representations of conversation flows (Y) are firstly rendered by a dialog simulator, and then realised into natural dialog (X) by annotators. Two central aspects which directly impact the quality of the resulting data are: (1) the dialog simulator which controls the coverage and naturalness of the conversations; (2) the annotation instructions and quality control mechanism that ensures X and Y are semantically aligned after annotation.

4.2 Simulator

The most common approach of simulating a conversation flow is agenda-based (Schatzmann et al., 2007; Li et al., 2016; Shah et al., 2018; Rastogi et al., 2019). At the beginning of this approach, a new goal is defined in the form of slot-value pairs describing user's requests and constraints; and an agenda is constructed by decomposing the user goal into a sequence of user actions. Although the approach ensures the user behaves in a goal-oriented manner, it constrains the output space with *pre-defined* agendas, which is hard to craft for complex user goals (Shi et al., 2019).

Arguably, a more natural solution to dialog simulation for complex output space is a fully generative method. It complies with the behavior that a real user may only have an initial goal at the start of conversation, while the final dialog state cannot be foreseen in advance. The whole conversation can be defined generatively as follows:

$$P(Y) = P(y_0^u) \sum_{t=0}^{n} P(y_t^s|y_t^u) \sum_{t=1}^{n} P(y_t^u|y_{<t}^s, y_{<t}^u) \tag{1}$$

where Y is the conversation flow, y_t^u is the user dialog state at turn t and y_t^s the system dialog act. The decomposed probability of $P(Y)$ captures the functional space of dialog state transitions with three components: 1) a module generating the initial user goal $P(y_0^u)$, 2) a module generating system act $P(y_t^s|y_t^u)$, and 3) a module for user state update based on the dialog history $P(y_t^u|y_{<t}^s, y_{<t}^u)$. The conversation terminates at time step n which must be a finishing state (system success or failure).

Initial intent module $P(y_0^u)$ The dialog state y_0^u representing the initial user goal is generated with a probabilistic tree substitution grammar (Cohn et al., 2010, PTSG) based on our semantic formalism. Non-terminal symbols in the PTSG can rewrite entire tree fragments to encode non-local context such as hierarchical and co-occurring slot combinations. As explained in the example below, the algorithm generates dialog states from top down by recursively substituting non-terminals (marked in green) with subtrees.

```
$createEvent  →user.calendarEvent.create
                    .$newEvent

$newEvent  →object.equals
                 .$attendees
                 .$location
```

This example generates a calendar event creation intent that contains two slots of attendees and location. In a statistical approach, the sampling probability of each production rule could be learned from dialog data. For the purpose of bootstrapping a new model, any prior distribution can be applied.

Response module $P(y_t^s|y_t^u)$ At turn t, the system act y_t^s is generated based on the current user dialog state y_t^u and domain-specific logic. We represent the probability $P(y_t^s|y_t^u)$ with a probabilistic tree transformation grammar, which captures how an output tree (y_t^s) is generated from the input (y_t^u) through a mixture of generation and a copy mechanism. As shown in the example below, every production rule in the grammar is in the form $A \rightarrow B$, where A is an input *pattern* that could contain both observed nodes and unobserved nodes (marked in red), and B is an output *pattern* to generate.

```
user.calendarEvent.create
     .object.equals
          . -dateTimeRange
                ↓
system.prompt.calendarEvent
     .create.object.equals
             .dateTimeRange
```

Given a user dialog state, the simulator *looks up* production rules which result in a match of pattern A, and then derives a system act based on the pattern B. Like in PTSG, probabilities of transformations can be either learned from data or specified a priori.

State update module $P(y_t^u | y_{<t}^s, y_{<t}^u)$ The generation of the updated dialog state is dependent on the dialog history. While the full space of y_t^u is unbounded, we focus on simulating three common types of updates, where a user introduces a new goal, continues with the previous goal y_{t-1}^u, or resumes an earlier unfinished goal (see Turn 2-4 of Table 1 for examples, respectively). To model dialog history, we introduce an *empty* stack when the conversation starts. A dialog state and the rendered system act are pushed onto the stack upon generation, dynamically updated during the conversation, and popped from the stack upon task completion. Therefore, the top of the stack always represents the most recent unfinished task $y_{t-1}^{top,u}$ and the corresponding system act $y_{t-1}^{top,s}$.

We consider the top elements of the stack as the effective dialog history and use it to generate the next dialog state y_t^u. The generation interface is modeled with a similar tree transformation grammar, but every production rule has two inputs in the form $A, B \to C$:

```
user.calendarEvent.create  system.prompt.calendarEvent
  .object.equals                 .create.object.equals
    . -dateTimeRange               .dateTimeRange
                          ↓

           user .calendarEvent.create
                .object.equals
                     . $dateTimeRange
```

where A specifies a matching pattern of the user goal $y_{t-1}^{top,u}$, B is a matching pattern of the system act $y_{t-1}^{top,s}$, and C is an output pattern that represents how the updated dialog state is obtained through a mixture of grammar expansion (marked in green) and copy mechanism from either of the two sources (marked in yellow).

4.3 Annotation

Following Shah et al. (2018); Rastogi et al. (2019), every grammar production in the simulator is paired with a template whose slots are synchronously expanded. As a result, each dialog state or system act is associated with a template utterance. The purpose is to offer minimum understandability to each conversation flow, based on which annotators will generate natural conversations.

Instructions Annotators generate a conversation based on the given templated utterances. The task proceeds turn by turn. For each turn, we instruct annotators to convey exactly the same intents and slots in each user or system utterance, in order to make sure the obtained utterance agrees with the programmatically generated semantic annotation. The set of open values (specially marked in brackets, such as event titles and text messages) must be preserved too. Besides the above restrictions, we give annotators the freedom to generate an utterance with paraphrasing, compression and expansion in the given dialog context, to make conversations as natural as possible. While system utterances are guided to be expressed in a professional tone, we encourage annotators to introduce adequate syntactic variations and chit-chats in user utterances as long as they do not change the intent to be delivered.

Quality control We enforce two quality control mechanisms before and after the dialog rendering task. Before the task, we ask annotators to provide a binary label to each conversation flow. The label indicates if the conversation contains any non-realistic interactions. We can therefore filter out low-quality data outputted by the simulator. After the task, we ask a different batch of annotators to evaluate if each human-generated utterance preserves the meaning of the templated utterance; any conversation that fails this check is removed.

4.4 Statistics

The resulting TreeDST dataset consists of 10 domains: *flight*, *train*, *hotel*, *restaurant*, *taxi*, *calendar*, *message*, *reminder*, *navigation*, and *phone*, exhibiting nested properties for *people*, *time*, and *location* that are shared across domains. Table 2 shows a comparison of TreeDST with the following pertinent datasets, **DSTC2** (Henderson

	DSTC2	WOZ2.0	FRAMES	M2M	MultiWOZ	SGD	TreeDST
Representation	Flat						Hierarchical
#Dialogs	1,612	600	1,369	1,500	8,438	16,142	27,280
Total #turns	23,354	4,472	19,986	14,796	113,556	329,964	167,507
Avg. #turns/dialog	14.5	7.45	14.60	9.86	13.46	20.44	7.14
Avg. #tokens/utterance	8.54	11.24	11.24	8.24	13.13	9.75	7.59
#slots	8	4	61	13	24	214	287
#values	212	99	3,871	138	4,510	14,139	20,612
#multi-domain dialog	-	-	-	-	7,032	16,142	14,999
#compositional utterance	-	-	-	-	-	-	10,133
#cross-turn co-reference	-	-	-	-	-	-	9,609

Table 2: Comparison of TreeDST with pertinent datasets for task-oriented dialogue.

et al., 2014a), **WOZ2.0** (Wen et al., 2017), **FRAMES** (El Asri et al., 2017), **M2M** (Shah et al., 2018), **MultiWOZ** (Budzianowski et al., 2018) and **SGD** (Rastogi et al., 2019). Similar to our work, both **M2M** and **SGD** use a simulator to generate conversation flows; and both **MultiWOZ** and **SGD** contain multi-domain conversations. The difference is that all the previous work represents dialog states as flat slot-value pairs, which are not able to capture complex relations such as compositional intents.

5 Dialog State Tracking

The objective in the conversational semantic parsing task is to predict the updated dialog state at each turn given the current user input and dialog history $P(y_t^u|x_t^u, X_{<t}, Y_{<t})$. We tackle the problem with an encoder-decoder model: at turn t, the model encodes the current user utterance x_t^u and dialog history, conditioned on which the decoder predicts the target dialog state y_t^u. We call this model the Tree Encoder-Decoder, or TED.

Dialog history When a dialog session involves task switching, there will be multiple, non-accumulable dialog states in the conversation history. Since it is expensive to encode the entire history $X_{<t}, Y_{<t}$ whose size grows with the conversation, we compute a fixed-size history representation derived from the previous conversation flow ($Y_{<t}$). Specifically, we reuse the notation of a stack to store past dialog states, and the top of the stack $y_{t-1}^{top,u}$ tracks the most recent uncompleted task. The dialog history is then represented with the previous dialog state y_{t-1}^u, the dialog state on top of the stack $y_{t-1}^{top,u}$, and the previous system dialog act y_{t-1}^s. We merge the two dialog states y_{t-1}^u and $y_{t-1}^{top,u}$ into a single tree Y_{t-1}^u for featurization.

Encoding We adopt three encoders for utterance x_t^u, system act y_{t-1}^s and dialog state Y_{t-1}^u respectively. For the user utterance x_t^u, a bidirectional LSTM encoder is used to convert the word sequence into an embedding list $\mathbf{H_x} = [\mathbf{h_1^x}, \mathbf{h_2^x}, \cdots, \mathbf{h_n^x}]$, where n is the length of the word sequence. For both the previous system act y_{t-1}^s and user state Y_{t-1}^u, we linearize them into strings through depth-first traversal (see Figure 1). Then the linearized y_{t-1}^s and Y_{t-1}^u are encoded with two separate bidirectional LSTMs. The outputs are two embedding lists: $\mathbf{H_s} = [\mathbf{h_1^s}, \mathbf{h_2^s}, \cdots, \mathbf{h_m^s}]$ where m is the length of the linearized system act sequence, and $\mathbf{H_u} = [\mathbf{h_1^u}, \mathbf{h_2^u}, \cdots, \mathbf{h_l^u}]$ where l is the length of the linearized dialog state sequence. The final outputs of encoding are $\mathbf{H_x}$, $\mathbf{H_s}$ and $\mathbf{H_u}$.

Decoding After encoding, the next dialog state y_t^u is generated with an LSTM decoder as a linearized string which captures the depth-first traversal of the target graph (see Figure 1).

At decoding step i, the decoder feeds the embedding of the previously generated token $y_{t,i-1}^u$ and updates the decoder LSTM state to $\mathbf{g_i}$:

$$\mathbf{g_i} = \text{LSTM}(\mathbf{g_{i-1}}, \mathbf{y_{t,i-1}^u}) \tag{2}$$

An attention vector is computed between the state $\mathbf{g_i}$ and each of the three encoder outputs $\mathbf{H^x}$, $\mathbf{H^s}$ and $\mathbf{H^u}$. For each of the encoder memory $\mathbf{H}$, the computation is defined as follows:

$$\begin{aligned} \mathbf{a_{i,j}} &= \textit{attn}(\mathbf{g_i}, \mathbf{H}) \\ \mathbf{w_{i,j}} &= \textit{softmax}(\mathbf{a_{i,j}}) \\ \bar{\mathbf{h}}_\mathbf{i} &= \sum_{j=1}^{n} \mathbf{w_{i,j}} \mathbf{h_j} \end{aligned} \tag{3}$$

where *attn* represents the feed-forward attention defined in Bahdanau et al. (2015) and the *softmax* is taken over index j. By applying the attention mechanism to all three sources, we get three

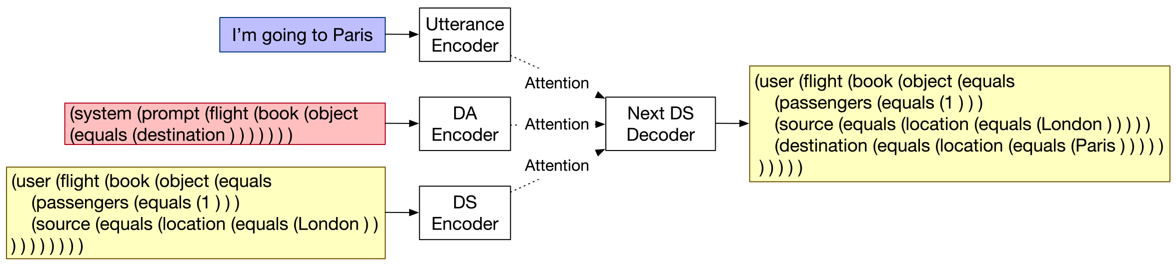
I'm going to Paris
Utterance Encoder
(system (prompt (flight (book (object (equals (destination)))))))
DA Encoder
(user (flight (book (object (equals (passengers (equals (1))) (source (equals (location (equals (London))))))))))
DS Encoder
Attention
Attention
Attention
Next DS Decoder
(user (flight (book (object (equals (passengers (equals (1))) (source (equals (location (equals (London))))) (destination (equals (location (equals (Paris))))))))))

Figure 1: An overview of the TED encoder-decoder architecture.

attention vectors $\bar{\mathbf{h}}^{\mathbf{x}}_{\mathbf{i}}$, $\bar{\mathbf{h}}^{\mathbf{s}}_{\mathbf{i}}$, and $\bar{\mathbf{h}}^{\mathbf{u}}_{\mathbf{i}}$. The vectors are concatenated together with the state $\mathbf{g_i}$ to form a feature vector $\mathbf{f_i}$, which is used to compute the probability of the next token though a mixture of generation and copy mechanism (Gu et al., 2016):

$$\begin{aligned}\lambda &= \sigma(\mathbf{W_i}\mathbf{f_i} + \mathbf{b_i}) \\ P_{gen} &= \mathit{softmax}(\mathbf{W_v}\mathbf{f_i} + \mathbf{b_v}) \\ P_{copy} &= \mathit{softmax}(\mathbf{a_i}, \mathbf{c_i}, \mathbf{e_i}) \\ P(y^u_{t,i}) &= \lambda P_{gen} + (1-\lambda)P_{copy}\end{aligned} \tag{4}$$

where $\mathbf{W}$ and $\mathbf{b}$ are all model parameters. λ is a soft gate controlling the proportion of generation and copy. P_{gen} is computed with a softmax over the generation vocabulary. $\mathbf{a}$, $\mathbf{c}$ and $\mathbf{e}$ denote attention logits computed for the three encoders. Since there are three input sources, we concatenate all logits and normalize them to compute the copy distribution P_{copy}. The model is optimised on the log-likelihood of output distribution $P(y^u_{t,i})$. An overview of the model is shown in Figure 1.

5.1 Parent Pointer: a Faster Graph Decoder

One observation about the standard decoder is that it has to predict long strings with closing brackets to represent a tree structure in the linearization. Therefore the total number of decoding LSTM recursions is the number of tree nodes plus the number of non-terminals. We propose a modified *parent pointer* (PP) decoder which reduces the number of autoregressions to the number of tree nodes. This optimisation is not applicable only to our DST model, but to any decoder that treats tree decoding as sequence prediction in the spirit of Vinyals et al. (2015).

The central idea of the PP decoder is that at each decoding step, two predictions will be made: one generates the next tree node, and the other selects its parent from the existing tree nodes. Eventually y^u_t can be constructed from a list of tree nodes n^u_t and a list of parent relations r^u_t. More specifically, at time step i, the decoder takes in the embeddings of the previous node $n^u_{t,i-1}$ and its parent $r^u_{t,i-1}$ to generate the hidden state g_i.

$$\mathbf{g_i} = \mathrm{LSTM}(\mathbf{g_{i-1}}, \mathbf{n^u_{t,i-1}}, \mathbf{r^u_{t,i-1}}) \tag{5}$$

This state is then used as as the input for two prediction layers. The first layer predicts the next node probability $P(n^u_{t,i})$ with Equation 3 to 4, and the second layer selects the parent of the node by attending $\mathbf{g_i}$ to the previously generated nodes, which are represented with the decoder memory $\mathbf{G_{i-1}} = [\mathbf{g_1}, \cdots, \mathbf{g_{i-1}}]$:

$$\begin{aligned}\mathbf{f_{i,j}} &= \mathit{attn}(\mathbf{g_i}, \mathbf{G_{i-1}}) \\ P(r^u_{t,i}) &= \mathit{softmax}(\mathbf{f_{i,j}})\end{aligned} \tag{6}$$

The model is optimised on the average negative log-likelihood of distributions $P(n^u_{t,i})$ and $P(r^u_{t,i})$.

6 Experiments

Setup We split the TreeDST data into train (19,808), test (3,739) and development (3,733) sets. For evaluation, we measure turn-level dialog state exact match accuracy averaged over all turns in the test set. We evaluate the proposed model with its "vanilla" decoder (TED-VANILLA) and its parent-pointer variant (TED-PP). In both cases, the utterance encoder has 2 layers of 500 dimensions; the system act encoder and dialog state encoder have 2 layers of 200 dimensions; and the decoder has 2 LSTM layers of 500 dimensions. Dimensions of word and tree node embeddings are 200 and 50 respectively. Training uses a batch size of 50 and Adam optimizer (Kingma and Ba, 2015). Validation is performed every 2 epochs and the training stops when the validation error does not decrease in four consecutive evaluations. The hyper-parameters were selected empirically based on an additional dataset that does not overlap with TreeDST.

Baselines In order to factor out the contribution of meaning representations from model changes in

Decoders	Accuracy
TED-VANILLA	0.622
TED-PP	0.622
TED-FLAT	0.535
COMER (Ren et al., 2019)	0.509
TRADE (Wu et al., 2019)	0.513

Table 3: Results on the TreeDST test set

Pattern	Accuracy
All turns	0.647
Turns with intent change	0.552
Turns with compositional utterances	0.602
Turns with multi-intent utterances	0.478

Table 4: Results on the TreeDST development set, broken down by dialog phenomena

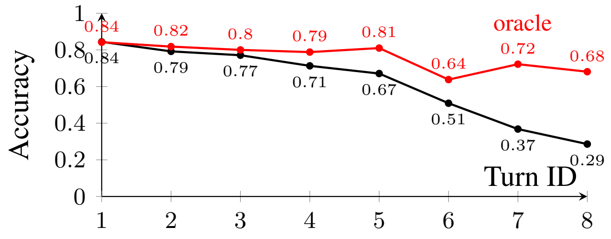

Figure 2: Validation exact match accuracy by turn ID

experiments, we additionally derive a version of our dataset where all meaning representations are flattened into slot-value pairs (details are described in the next paragraph). We then introduce a baseline TED-FLAT by training the same model (as TED-VANILLA) on the flattened dataset.

We also introduce as baselines two state-of-the-art slot-filling DST models based on encoder-decoders: they include COMER (Ren et al., 2019) which encodes the previous system response transcription and the previous user dialog state and decodes slot values; and TRADE (Wu et al., 2019) which encodes all utterances in the history. Since both TED-FLAT and the two baselines are trained with flattened slot-value representations, we can compare various models in this setup.

TreeDST flattening To flatten TreeDST we collapse each path from the `domain` to leaf nodes into a single slot. `Verb` nodes in the path are excluded to avoid slot explosion. Take the following tree as an example:

```
user.flight.book.object.equals
  .source.equals.location.equals.London
  .departureDateTime.equals
    .date.equals.definedValue.equals
      .Tomorrow
    .time.equals.hour.equals.5
```

Three slot-value pairs can be extracted:

```
(flight+object+source+location, London)
(flight+object+departureDateTime+date
+definedValue, Tomorrow)
(flight+object+departureDateTime+time
+hour, 5)
```

The operator `equals` is not shown in the slot names to make the names more concise.

Results The results are shown in Table 3. Overall, both TED-VANILLA and TED-PP models bring 20% relative improvement over existing slot-filling-based DST models. By further factoring out the impact of representation and model differences, we see that representation plays a more important role: the TED-FLAT variant, which differs only that it was trained on flattened parses, is clearly outperformed by TED-VANILLA and TED-PP. We conclude that even if dialog states can be flattened into slot-value pairs, it is still more favorable to use a compact, hierarchical meaning representation. The advantage of the representation is that it improves knowledge sharing across different domains (e.g., `message` and `phone`), verbs (e.g., `create` and `update`), and dialog participators (`user` and `system`). The second set of comparison is among different modeling approaches using the same flat meaning representation. TED-FLAT slightly outperforms COMER and TRADE. The major difference is that our model encodes both past user and system representations; while the other models used past transcriptions. We believe the gain of encoding representations is that they are unambiguous; and the encoding helps knowledge sharing between the user and the system.

The vanilla and PP decoders achieve the same exact match accuracy. The average training time per epoch for PP is 1,794 seconds compared to 2,021 for vanilla, i.e. PP leads to 10% reduction in decoding time without any effect on accuracy. We believe that the efficiency of PP can be further optimized by parallellizing the two prediction layers of nodes and parents.

Analysis First, Table 4 shows a breakdown of the TED-PP model performance by dialog behavior on the development set. While the model does fairly well on compositional utterances, states with intent switching and multiple intents are harder to predict. We believe the reason is that the prediction of intent switching requires task reference resolution within the model; while multi-intent utterances tend to have more complex trees.

Second, Figure 2 shows the vanilla model results by turn index on the development set (black curve). This shows the impact of error propagation as the model predicts the target dialog state based on past representations. To better understand the issue, we compare to an oracle model which always uses the gold previous dialog state for encoding (red curve). Vanilla model accuracy decreases with turn index, resulting in a gap with the oracle model. The error propagation problem can be alleviated by providing more complete dialog history to the encoder for error recovery (Henderson et al., 2014b), which we consider as future work.

Finally, we would like to point out a limitation of our approach in tracking dialog history with a stack-based memory. While the stack is capable of memorizing and returning to a previously unfinished task, there are patterns which cannot be represented such as switching between two ongoing tasks. We aim to explore a richer data structure for dialog history in the future.

7 Conclusion

This work reformulates dialog state tracking as a conversational semantic parsing task to overcome the limitations of slot filling. Dialog states are represented as rooted relational graphs to encode compositionality, and encourage knowledge sharing across different domains, verbs, slot types and dialog participators. We demonstrated how a dialog dataset with structured labels for both user and system utterances can be collected with the aid of a generative dialog simulator. We then proposed a conversational semantic parser that performs DST with an encoder-decoder model and a stack-based memory. A parent-pointer decoder is further proposed to speed up tree prediction. Experimental results show that our DST solution outperforms slot-filling-based trackers by a large margin.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*.

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

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026.

Bill Byrne, Karthik Krishnamoorthi, Chinnadhurai Sankar, Arvind Neelakantan, Ben Goodrich, Daniel Duckworth, Semih Yavuz, Amit Dubey, Kyu-Young Kim, and Andy Cedilnik. 2019. Taskmaster-1: Toward a realistic and diverse dialog dataset. 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 4506–4517.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11(Nov):3053–3096.

Paul A Crook and Oliver Lemon. 2010. Representing uncertainty about complex user goals in statistical dialogue systems. In *Proceedings of the SIGDIAL 2010 Conference*, pages 209–212.

Layla El Asri, Hannes Schulz, Shikhar Kr Sarma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. 2017. Frames: a corpus for adding memory to goal-oriented dialogue systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 207–219.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Matthew Henderson, Blaise Thomson, and Jason D. Williams. 2014a. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272, Philadelphia, PA, U.S.A. Association for Computational Linguistics.

Matthew Henderson, Blaise Thomson, and Steve Young. 2014b. Word-based dialog state tracking with recurrent neural networks. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 292–299.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations*.

Thomas Kollar, Danielle Berry, Lauren Stuart, Karolina Owczarzak, Tagyoung Chung, Lambert Mathias, Michael Kayser, Branrd Snow, and Spyros Matsoukas. 2018. The alexa meaning representation language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 177–184.

Hwaran Lee, Jinsik Lee, and Tae-Yoon Kim. 2019. Sumbt: Slot-utterance matching for universal and scalable belief tracking. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5478–5483.

Sungjin Lee and Amanda Stent. 2016. Task lineages: Dialog state tracking for flexible interaction. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 11–21.

Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. 2016. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788.

Nikola Mrkšić and Ivan Vulić. 2018. Fully statistical neural belief tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 108–113.

Denis Peskov, Nancy Clarke, Jason Krone, Brigi Fodor, Yi Zhang, Adel Youssef, and Mona Diab. 2019. Multi-domain goal-oriented dialogues (multidogo): Strategies toward curating and annotating large scale dialogue data. 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 4518–4528.

Filip Radlinski, Krisztian Balog, Bill Byrne, and Karthik Krishnamoorthi. 2019. Coached conversational preference elicitation: A case study in understanding movie preferences. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 353–360.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2019. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *arXiv preprint arXiv:1909.05855*.

Liliang Ren, Jianmo Ni, and Julian McAuley. 2019. Scalable and accurate dialogue state tracking via hierarchical sequence generation. 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 1876–1885.

Alexander Rudnicky and Wei Xu. 1999. An agenda-based dialog management architecture for spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Workshop*.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. 2018. Building a conversational agent overnight with dialogue self-play. *arXiv preprint arXiv:1801.04871*.

Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. 2019. How to build user simulators to train rl-based dialog systems. 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 1990–2000.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proceedings of NIPS 2015*.

Tsung Hsien Wen, David Vandyke, Nikola Mrki, Milica Gasic, and Steve Young. 2017. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*.

Jason Williams, Antoine Raux, and Matthew Henderson. 2016. The dialog state tracking challenge series: A review. *Dialogue & Discourse*, 7(3):4–33.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 808–819.

Puyang Xu and Qi Hu. 2018. An end-to-end approach for handling unknown slot values in dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1448–1457.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

A Appendix. Dotted Tree

A dotted tree format is used throughout the paper to reduce the space of drawing and the mental space of readers. Dots represent edges between two tree nodes. When a node has multiple children attached to it, indentation is applied to reveal the hierarchy.

The following representation is the dotted format for the tree in Figure 3.

```
user.flight.book.object.equals
  .source.equals.location.equals.London
  .destination.equals.location.equals.Paris
  .departureDateTime.equals
    .date.equals.definedValue.equals
      .Tomorrow
    .time.equals
      .hour.equals.10
      .meridiem.equals.AM
```

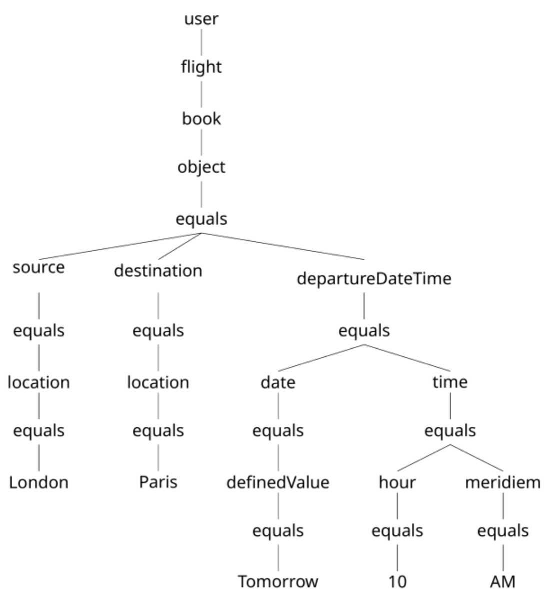

Figure 3: A tree representing the user intent *Book a ticket from London to Paris tomorrow 10 AM*.