In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15), pp. 878--888, Beijing, China, July 2015.

Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes

Chris Quirk
Microsoft Research
Redmond, WA, USA
chrisq@microsoft.com

Raymond Mooney*
UT Austin
Austin TX, USA
mooney@cs.utexas.edu

Michel Galley
Microsoft Research
Redmond, WA, USA
mgalley@microsoft.com

Abstract

Using natural language to write programs is a touchstone problem for computational linguistics. We present an approach that learns to map natural-language descriptions of simple "if-then" rules to executable code. By training and testing on a large corpus of naturally-occurring programs (called "recipes") and their natural language descriptions, we demonstrate the ability to effectively map language to code. We compare a number of semantic parsing approaches on the highly noisy training data collected from ordinary users, and find that loosely synchronous systems perform best.

1 Introduction

The ability to program computers using natural language would clearly allow novice users to more effectively utilize modern information technology. Work in *semantic parsing* has explored mapping natural language to some formal domain-specific programming languages such as database queries (Woods, 1977; Zelle and Mooney, 1996; Berant et al., 2013), commands to robots (Kate et al., 2005), operating systems (Branavan et al., 2009), smartphones (Le et al., 2013), and spreadsheets (Gulwani and Marron, 2014). Developing such language-to-code translators has generally required specific dedicated efforts to manually construct parsers or large corpora of suitable training examples.

An interesting subset of the possible program space is if-then "recipes," simple rules that allow users to control many aspects of their digital life including smart devices. Automatically parsing these recipes represents a step toward complex natural language programming, moving beyond single commands toward compositional statements with control flow.

Several services, such as Tasker and IFTTT, allow users to create simple programs with "triggers" and "actions." For example, one can program their Phillips Hue light bulbs to flash red and blue when the Cubs hit a home run. A somewhat complicated GUI allows users to construct these recipes based on a set of information "channels." These channels represent many types of information. Weather, news, and financial services have provided constant updates through web services. Home automation sensors and controllers such as motion detectors, thermostats, location sensors, garage door openers, etc. are also available. Users can then describe the recipes they have constructed in natural language and publish them.

Our goal is to build semantic parsers that allow users to describe recipes in natural language and have them automatically mapped to executable code. We have collected 114,408 recipe-description pairs from the http://ifttt.com website. Because users often provided short or incomplete English descriptions, the resulting data is extremely noisy for the task of training a semantic parser. Therefore, we have constructed semantic-parser learners that utilize and adapt ideas from several previous approaches (Kate and Mooney, 2006; Wong and Mooney, 2006) to learn an effective interpreter from such noisy training data. We present results on our collected IFTTT corpus demonstrating that our best approach produces more accurate programs than several competing baselines. By exploiting such "found data" on the web, semantic parsers for natural-language programming can potentially be developed with minimal effort.

*Work performed while visiting Microsoft Research.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 878–888, Beijing, China, July 26-31, 2015. ©2015 Association for Computational Linguistics

2 Background

We take an approach to semantic parsing that directly exploits the formal grammar of the target meaning representation language, in our case IFTTT recipes. Given supervised training data in the form of natural-language sentences each paired with their corresponding IFTTT recipe, we learn to introduce productions from the formal-language grammar into the derivation of the target program based on expressions in the natural-language input. This approach originated with the SILT system (Kate et al., 2005) and was further developed in the WASP (Wong and Mooney, 2006; Wong and Mooney, 2007b) and KRISP (Kate and Mooney, 2006) systems.

WASP casts semantic parsing as a syntax-based *statistical machine translation* (SMT) task, where a *synchronous context-free grammar* (SCFG) (Wu, 1997; Chiang, 2005; Galley et al., 2006) is used to model the translation of natural language into a formal meaning representation. It uses statistical models developed for syntax-based SMT for lexical learning and parse disambiguation. Productions in the formal-language grammar are used to construct synchronous rules that simultaneously model the generation of the natural language. WASP was subsequently "inverted" to use the same synchronous grammar to generate natural language from the formal language (Wong and Mooney, 2007a).

KRISP uses classifiers trained using a Support-Vector Machine (SVM) to introduce productions in the derivation of the formal translation. The productions of the formal-language grammar are treated like semantic concepts to be recognized from natural-language expressions. For each production, an SVM classifier is trained using a *string subsequence kernel* (Lodhi et al., 2002). Each classifier can then estimate the probability that a given natural-language substring introduces a production into the derivation of the target representation. During semantic parsing, these classifiers are employed to estimate probabilities on different substrings of the sentence to compositionally build the most probable meaning representation for the sentence. Unlike WASP whose synchronous grammar needs to be able to directly parse the input, KRISP's approach to "soft matching" productions allows it to produce a parse for *any* input sentence. Consequently, KRISP was shown to be much more robust to noisy training data than previous approaches to semantic parsing (Kate and Mooney, 2006).

Since our "found data" for IFTTT is extremely noisy, we have taken an approach similar to KRISP; however, we use a probabilistic log-linear text classifier rather than an SVM to recognize productions.

This method of assembling well-formed programs guided by a natural language query bears some resemblance to Keyword Programming (Little and Miller, 2007). In that approach, users enter natural language queries in the middle of an existing program; this query drives a search for programs that are relevant to the query and fit within the surrounding program. However, the function used to score derivations is a simple matching heuristic relying on the overlap between query terms and program identifiers. Our approach uses machine learning to build a correspondence between queries and recipes based on parallel data.

There is also a large body of work applying Combinatory Categorical Grammars to semantic parsing, starting with Zettlemoyer and Collins (2005). Depending on the set of combinators used, this approach can capture more expressive languages than synchronous context-free MT. In practice, however, synchronous MT systems have competitive accuracy scores (Andreas et al., 2013). Therefore, we have not yet evaluated CCG on this task.

3 If-this-then-that recipes

The recipes considered in this paper are diverse and powerful despite being simple in structure. Each recipe always contains exactly one trigger and one action. Whenever the conditions of the trigger are satisfied, the action is performed. The resulting recipes can perform tasks such as home automation ("turn on my lights when I arrive home"), home security ("text me if the door opens"), organization ("add receipt emails to a spreadsheet"), and much more ("remind me to drink water if I've been at a bar for more than two hours"). Triggers and actions are drawn from a wide range of channels that must be activated by each user. These channels can represent many entities and services, including devices (such as Android devices or WeMo light switches) and knowledge sources (such as ESPN or Gmail). Each channel exposes a set of functions for both trigger and action.

Several services such as IFTTT, Tasker, and Llama allow users to author if-this-then-that recipes. IFTTT is unique in that it hosts a large set of recipes along with descriptions and other metadata. Users of this site construct recipes using a GUI interface to select the trigger, action, and the

parameters for both trigger and action. After the recipe is authored, the user must provide a description and optional set of notes for this recipe and publish the recipe. Other users can browse and use these published recipes; if a user particularly likes a recipe, they can mark it as a favorite.

As of January 2015, we found 114,408 recipes on http://ifttt.com. Among the available recipes we encountered a total of 160 channels. In total, we found 552 trigger functions from 128 of those channels, and 229 action functions from 99 channels, for a total of 781 functions. Each recipe includes a number of pieces of information: description[1], note, author, number of uses, etc. 99.98% of the entries have a description, and 35% contain a note. Based on availability, we focused primarily on the description, though there are cases where the note is a more explicit representation of program intent.

The recipes at http://ifttt.com are represented as HTML forms, with combo boxes, inline maps, and other HTML UI components allowing end users to select functions and their parameters. This is convenient for end users, but difficult for automated approaches. We constructed a *formal grammar* of possible program structures, and from each HTML form we extracted an abstract syntax tree (AST) conforming to this grammar. We model this as a context-free grammar, though this assumption is violated in some cases. Consider the program in Figure 1, where some of the parameters used the action are provided by the trigger.

This data could be used in a variety of ways. Recipes could be suggested to users based on their activities or interests, for instance, or one could train a natural language generation system to give a readable description of code.

In this paper, the paired natural language descriptions and abstract syntax trees serve as training data for semantic parsing. Given a description, a system must produce the AST for an IFTTT recipe. We note in passing that the data was constructed in the opposite direction: users first implemented the recipe and then provided a description afterwards. Ideal data for our application would instead start with the description and construct the recipe based on this description. Yet the data is unusually large and diverse, making it interesting training data for mapping natural language to code.

[1]The IFTTT site refers to this as "title".

4 Program synthesis methods

We consider a number of methods to map the natural language description of a problem into its formal program representation.

4.1 Program retrieval

One natural baseline is retrieval. Multiple users could potentially have similar needs and therefore author similar or even identical programs. Given a novel description, we can search for the closest description in a table of program-description pairs, and return the associated program. We explored several text-similarity metrics, and found that string edit distance over the unmodified character sequence achieved best performance on the development set. As the corpus of program-description pairs becomes larger, this baseline should increase in quality and coverage.

4.2 Machine Translation

The downside to retrieval is that it cannot generalize. Phrase-based SMT systems(Och et al., 1999; Koehn et al., 2003) can be seen as an incremental step beyond retrieval: they segment the training data and attempt to match and assemble those segments at runtime. If the phrase length is unbounded, retrieval is almost a special case: it could return whole programs from the training data when the description matches exactly. In addition, they can find subprograms that are relevant to portions of the input, and assemble those subprograms into whole programs.

As a baseline, we adopt a recent approach (Andreas et al., 2013) that casts semantic parsing as phrasal translation. First, the ASTs are converted into flat sequences of code tokens using a pre-order left-to-right traversal. The tokens are annotated with their arity, which is sufficient to reconstruct the tree given a well formed sequence of tokens using a simple stack algorithm. Given this parallel corpus of language and code tokens, we train a conventional statistical machine translation system that is similar in structure and performance to Moses (Koehn et al., 2007). We gather the k-best translations, retaining the first such output that can be successfully converted into a well-formed program according to the formal grammar. Integration of the well-formedness constraint into decoding would likely produce better translations, but would require more modifications to the MT system.

Approaches to semantic parsing inspired by machine translation have proven effective when the

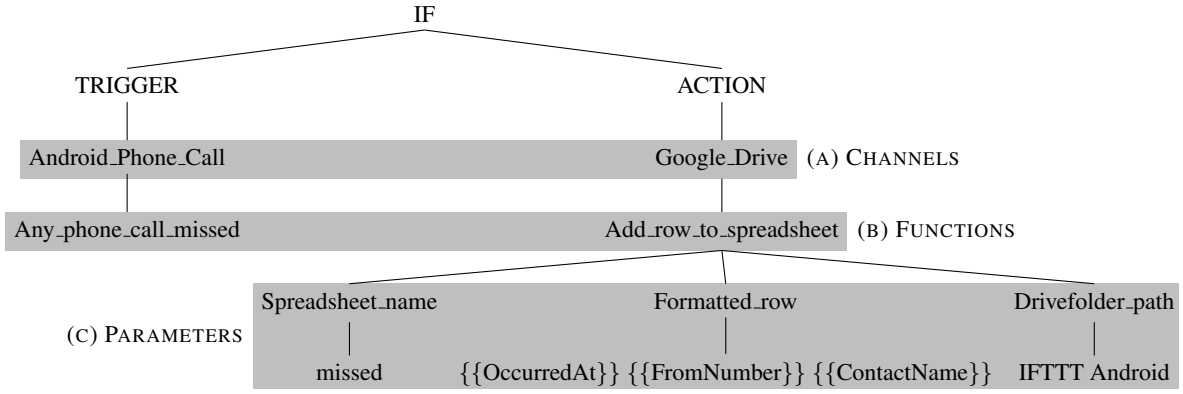

Figure 1: Example recipe with description, with nodes corresponding to (a) Channels, (b) Functions, and (c) Parameters indicated with specific boxes. Note how some of the fields in braces, such as OccurredAt, depend on the trigger.

data is very parallel. In the IFTTT dataset, however, the available pairs are not particularly clean. Word alignment quality suffers, and production extraction suffers in turn. Descriptions in this corpus are often quite telegraphic (e.g., "Instagram to Facebook") or express unnecessary pieces of information, or are downright unintelligible ("_2Mrl14"). Approaches that rely heavily on lexicalized information and assume a one-to-one correspondence between source and target (at the phrase, if not the word level) struggle in this setting.

4.3 Generation without alignment

An alternate approach is to treat the source language as context and a general direction, rather than a hard constraint. The target derivation can be produced primarily according to the formal grammar while guided by features from the source language.

For each production in the formal grammar, we can train a binary classifier intended to predict whether that production should be present in the derivation. This classifier uses general features of the source sentence. Note how this allows productions to be inferred based on context: although a description might never explicitly say that a production is necessary, the surrounding context might strongly imply it.

We assign probabilities to derivations by looking at each production independently. A derivation either uses or does not use each production. For each production used in the derivation, we multiply by the probability of its inclusion. Likewise for each production *not* used in the derivation, we multiply by one minus the probability of its inclusion.

Let $G = (V, \Sigma, R, S)$ be the formal grammar with non-terminals V, terminal vocabulary Σ, productions R and start symbol S. E represents a source sentence, and D, a formal derivation tree for that sentence. $R(D)$ is the set of productions in that derivation. The score of a derivation is the following product:

$$P(D|E) = \prod_{r \in R(D)} P(r|E) \prod_{r \in R \setminus R(D)} P(\neg r|E)$$

The binary classifiers are log-linear models over features, F, of the input string: $P(r|E) \propto \exp\left(\theta_r^\top F(E)\right)$.

4.3.1 Training

For each production, we train a binary classifier predicting its presence or absence. Given a training set of parallel descriptions and programs, we create $|R|$ binary classifier training sets, one for each classifier. We currently use a small set of simple features: word unigrams and bigrams, and character trigrams.

4.3.2 Inference

When presented with a novel utterance, E, our system must find the best code corresponding to that utterance. We use a top-down, left-to-right generation strategy, where each search node contains a stack of symbols yet to be expanded and a log probability. The initial node is $\langle [S], 0 \rangle$; and a node is complete when its stack of non-terminals is empty.

Given a search node with a non-terminal as its first symbol on the stack, we expand with any production for that symbol, putting its yield onto the stack and updating the node cost to include its

derivation score:

$$\frac{\langle [X, \alpha], p \rangle \;\; (X \rightarrow \beta) \in R}{\langle [\beta, \alpha], p + \log P(X \rightarrow \beta | E) \rangle}$$

If the first stack item is a terminal, it is scanned:

$$\frac{\langle [a, \alpha], p \rangle \;\; a \in \Sigma}{\langle [\alpha], p \rangle}$$

Using these inference rules, we utilize a simple greedy approach that only accounts for the productions included in the derivation. To account for the negative $\prod_{r \in R \setminus R(D)} P(\neg r|E)$ factors, we use a beam search, and rerank the n-best final outcomes from this search based on the probability of all productions that are not included. Partial derivations are grouped into beams according to the number of productions in that derivation.

4.4 Loosely synchronous generation

The above method learns distributions over productions given the input, but treats the sentence as an undifferentiated bag of linguistic features. The syntax of the source sentence is not leveraged at all, nor is any correspondence between the language syntax and the program structure used. Often the pairs are not in sufficient correspondence to suggest synchronous approaches, but some loose correspondence to maintain at least a notion of coverage could be helpful.

We pursue an approach similar to KRISP (Kate and Mooney, 2006), with several differences. First, rather than a string kernel SVM, we use a log-linear model with character and word n-gram features. [2] Second, we allow the model to consider both span-internal features and contextual features.

This approach explicitly models the correspondence between nodes in the code side and tokens in the language. Unlike standard MT systems, word alignment is not used as a hard constraint. Instead, this phrasal correspondence is induced as part of model training.

We define a semantic derivation D of a natural language sentence E as a program AST where each production in the AST is augmented with a span. The substrings covered by the children of a production must not overlap, and the substring covered by the parent must be the concatenation of the substrings covered by the children. Figure 2 shows a sample semantic derivation.

[2] We have a preference for log-linear models given their robustness to hyperparameter settings, ease of optimization, and flexible incorporation of features. An SVM trained with similar features should have similar performance, though.

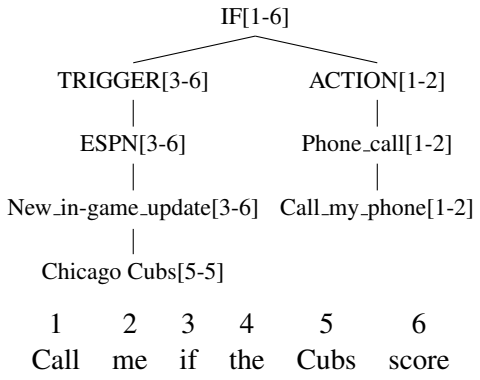

Figure 2: An example training pair with its semantic derivation. Note the correspondence between formal language and natural language denoted with indices and spans.

The core components of KRISP are string-kernel classifiers $P(r, i..j|E)$ denoting the probability that a production r in the AST covers the span of words $i..j$ in the sentence E. Here, $i < j$ are positions in the sentence indicating the span of tokens most relevant to this production. In other words, the substring $E[i..j]$ denotes the production r with probability $P(r, i..j|E)$. The probability of a semantic derivation D is defined as follows:

$$P(D|E) = \prod_{(r,i..j) \in D} P(r, i..j|E)$$

That is, we assume that each production is independent of all others, and is conditioned only on the string to which it is aligned. This can be seen as a refinement of the above production classification approach using a notion of correspondence.

Rather than using string kernels, we use logistic regression classifiers with word unigram, word bigram, and character trigram features. Unlike KRISP, we include features from both inside and outside the substring. Consider the production "Phone_call → Call_my_phone" with span 1-2 from Figure 2. Word unigram features indicate that "call" and "me" are inside the span; the remaining words are outside the span. Word bigram features indicate that "call me" is inside the span, "me if" is on the boundary of the span, and all remaining bigrams are outside the span.

4.4.1 Training

These classifiers are trained in an iterative EM-like manner (Kate and Mooney, 2006). Starting with some initial classifiers and a training set of NL and AST pairs, we search for the most likely derivation. If the AST underlying this derivation matches the gold AST, then this derivation is added

to the set of positive instances. Otherwise, it is added to the set of negative instances, and the best derivation constrained to match the gold standard AST is found and added to the positive instances. Given this revised training data, the classifiers are retrained. After each pass through the training data, we evaluate the current model on the development set. This procedure is repeated until development-set performance begins to fall.

4.4.2 Inference

To find the most probable derivation according to the grammar, KRISP uses a variation on Earley parsing. This is similar to the inference method from Section 4.3.2, but each item now additionally maintains a position and a span. Inference proceeds left-to-right through the source string. The natural language may present information in a different order than the formal language, so all permutations of rules are considered during inference.

We found this inference procedure to be quite slow for larger data sets, especially because wide beams were needed to prevent search failure. To speed up inference, we used scores from the position-independent classifiers as completion-cost estimates.

The completion-cost estimate for a given symbol is defined recursively. Terminals have a cost of zero. Productions have a completion cost of the log probability of the production given the sentence, plus the completion cost of all non-terminal symbols. The completion cost for a non-terminal is the max cost of any production rooted in that non-terminal. Computing this cost requires traversing all productions in the grammar for each sentence.

Given a partial hypothesis, we use exact scores for the left-corner subtree that has been fully constructed, and completion estimates for all the symbols and productions whose left and right spans are not yet fully instantiated.

5 Experimental Evaluation

Next we evaluate the accuracy of these approaches. The 114,408 recipes described in Section 3 were first cleaned and tokenized. We kept only one recipe per unique description, after mapping to lowercase and normalizing punctuation.[3] Finally the recipes were split by author, randomly assigning each to training, development, or test, to prevent overfitting to the linguistic style of a particular author. Table 1 presents summary statistics for the resulting data.

		Language	Code
Train	Recipes	77,495	77,495
	Tokens	527,368	1,776,010
	Vocabulary	58,102	140,871
Dev	Recipes	5,171	5,171
	Tokens	37,541	110,074
	Vocabulary	7,741	14,804
Test	Recipes	4,294	4,294
	Tokens	28,214	94,367
	Vocabulary	6,782	13,969

Table 1: Statistics of the data after cleaning and separating into training, development, and test sets. In each case, the number of recipes, tokens (including punctuation, etc.) and vocabulary size are included.

Although certain trigger-action pairs occur much more often than others, the recipes in this data are quite diverse. The top 10 trigger-action pairs account for 14% of the recipes; the top 100 account for 37%; the top 1000 account for 72%.

5.1 Metrics

To evaluate system performance, several different measures are employed. Ideally a system would output exactly the correct abstract syntax tree. One measure is to count the number of exact matches, though almost all methods receive a score of 0.[4]

Alternatively, we can look at the AST as a set of productions, computing balanced F-measure. This is a much more forgiving measure, giving partial credit for partially correct results, though it has the caveat that all errors are counted equally.

Correctly assigning the trigger and action is the most important, especially because some of the parameter values are tailored for particular users. For example, "turn off my lights when I leave home" requires a "home" location, which varies for each user. Therefore, we also measure accuracy at identifying the correct trigger and action, both at the channel and function level.

5.2 Human comparison

One remaining difficulty is that multiple programs may be equally correct. Some descriptions are very difficult to interpret, even for humans. Second,

[3] We found many recipes with the same description, likely copies of some initial recipe made by different users. We selected one representative using a deterministic heuristic.

[4] Retrieval gets an exact match 3.7% of the time, likely due to near-duplicates from copied recipes.

multiple channels may provide similar functionality: both Phillips Hue and WeMo channels provide the ability to turn on lights. Even a well-authored description may not clarify which channel should be used. Finally, many descriptions are underspecified. For instance, the description "notify me if it rains" does not specify whether the user should receive an Android notification, an iOS notification, an email, or an SMS. This is difficult to capture with an automatic metric.

To address the prevalence and impact of underspecification and ambiguity in descriptions, we asked humans to perform a very similar task. Human annotators on Amazon Mechanical Turk ("turkers") were presented with recipe descriptions and asked to identify the correct channel and function (but not parameters). Turkers received careful instructions and several sample description-recipe pairs, then were asked to specify the best recipe for each input. We requested they try their best to find an action and a trigger even when presented with vague or ambiguous descriptions, but they could tag inputs as 'unintelligible' if they were unable to make an educated guess. Turkers created recipes only for English descriptions, applying the label 'non-English' otherwise. Five recipes were gathered for each description. The resulting recipes are not exactly gold, as they have limited training at the task. However, we imposed stringent qualification requirements to control the annotation quality.[5]

Our workers were in fair agreement with one another and the gold standard, producing high quality annotation at wages calibrated to local minimum wage. We measure turker agreement with Krippendorff's α (Krippendorff, 1980), which is a statistical measure of agreement between any number of coders. Unlike Cohen's κ (Cohen, 1960), the α statistic does not require that coders be the same for each unit of analysis. This property is particularly desirable in our case, since turkers generally differ across HITs. A value of $\alpha = 1$ indicates perfect agreement, while $\alpha \leq 0$ suggests the absence of agreement or systematic disagreement. Agreement measures on the Mechanical Turk data are shown in Table 2. This shows encouraging levels of agreement for both the trigger and the action, especially considering the large number of categories. Krippendorff (1980) advocates a 0.67 cutoff to allow

[5]Turkers must have 95% HIT approval rating and be native speakers of English (As an approximation of the latter, we required Turkers be from the U.S.). Manual inspection of annotation on a control set drawn from the training data ensured there was no apparent spam.

	Trigger		Action	
	C	C+F	C	C+F
# of categories	*128*	*552*	*99*	*229*
all	.592	.492	.596	.532
Intelligible English	.687	.528	.731	.627

Table 2: Annotator agreement as measured by Krippendorff's α coefficient (Krippendorff, 1980). Agreement is measured on either channel (C) or channel and function (C+F), and on either the full test set (4294 recipes) or its English and intelligible subset (2262 recipes).

"tentative conclusion" of agreement, and turkers are relatively close to that level for both trigger and action channels. However, it is important to note that the coding scheme used by turkers is not mutually exclusive, as several triggers and actions (e.g., "SMS" vs. "Android SMS" actions) accomplish similar effects. Thus, our levels of agreement are likely to be greater than suggested by measures in the table. Finally, we also measured agreement on the English and intelligible subset of the data, as we found that confusion between the two labels "non-English" and "unintelligible" was relatively high. As shown in the table, this substantially increased levels of agreement, up to the point where α for both trigger and action channels are above the 0.67 cutoff drawing tentative conclusion of agreement.

5.3 Systems and baselines

The **retrieval** method searches for the closest description in the training data based on character string-edit-distance and returns the recipe for that training program. The **phrasal** method uses phrase-based machine translation to generate candidate outputs, searching the resulting n-best candidates for the first well-formed recipe. After exploring multiple word alignment approaches, we found that an unsupervised feature-rich method (Berg-Kirkpatrick et al., 2010) worked best, leveraging features of string similarity between the description and the code. We ran MERT on the development data to tune parameters. We used a phrasal decoder with performance similar to Moses. The **sync**hronous grammar method, a recreation of WASP, uses the same word alignment as above, but extracts a synchronous grammar rules from the parallel data (Wong and Mooney, 2006). The **classifier** approach described in Section 4.3 is independent of word alignment. Finally, the **posclass** approach from Section 4.4 derives its own deriva-

tion structure from the data.

The human annotations are used to establish the **mturk** human-performance baseline by taking the majority selection of the trigger and action over 5 HITs for each description and comparing the result to the gold standard. The **oracleturk** human-performance baseline shows how often at least one of the turkers agreed with the gold standard.

In addition, we evaluated all systems on a subset of the test data where at least three human-generated recipes agreed with the gold standard. This subset represents those programs that are easily reproducible by human workers. A good method should strive to achieve 100% accuracy on this set, and we should perhaps not be overly concerned about the remaining examples where humans disagree about the correct interpretation.

5.4 Results and discussion

Table 3 summarizes the main evaluation results. Most of the measures are in concordance.

Interestingly, retrieval outperforms the phrasal MT baseline. With a sufficiently long phrase limit, phrasal MT approaches retrieval, but with a few crucial differences. First, phrasal requires an exact match of some substring of the input to some substring of the training data, where retrieval can skip over words. Second, the phrases are heavily dependent on word alignment; we find the word alignment techniques struggle with the noisy IFTTT descriptions. Sync performs similarly to phrasal. The underspecified descriptions challenge assumptions in synchronous grammars: much of the target structure is implied rather than stated.

In contrast, the classification method performs quite well. Some productions may be very likely given a prior alone, or may be inferred given other productions and the need for a well-formed derivation. Augmenting this information with positional information as in posclass can help with the attribution problem. Consider the input "Download Facebook Photos you're tagged in to Dropbox": we would like the token "Facebook" to invoke only the trigger, not the action. We believe further gains could come from better modeling of the correspondence between derivation and natural language.

We find that semantic parsing systems have accuracy nearly as high or even higher than turkers in certain conditions. There are several reasons for this. First, many of the channels overlap in functionality (Gmail vs. email, or Android SMS vs. SMS); likewise functions may be very closely related (Post a tweet vs. Post a tweet with an image). All the systems with access to thousands of training pairs are at a strong advantage; they can, for

	Channel	+Func	Prod F1
(a) All: 4,294 recipes			
retrieval	28.2	19.3	40.8
phrasal	17.3	10.0	34.8
sync	16.2	9.5	34.9
classifier	46.3	33.0	47.3
posclass	**47.4**	**34.5**	**48.0**
mturk	33.4	22.6	–n/a–
oracleturk	48.8	37.8	–n/a–
(b) Omit non-English: 3,741 recipes			
retrieval	28.9	20.2	41.7
phrasal	19.3	11.3	35.3
sync	18.1	10.6	35.1
classifier	48.8	35.2	48.4
posclass	**50.0**	**36.9**	**49.3**
mturk	38.4	26.0	–n/a–
oracleturk	56.0	43.5	–n/a–
(c) Omit non-English & unintelligible: 2,262 recipes			
retrieval	36.8	25.4	49.0
phrasal	27.8	16.4	39.9
sync	26.7	15.5	37.6
classifier	64.8	47.2	56.5
posclass	**67.2**	**50.4**	**57.7**
mturk	59.0	41.5	–n/a–
oracleturk	86.2	59.4	–n/a–
(d) $\geq$3 turkers agree with gold: 758 recipes			
retrieval	43.3	32.3	56.2
phrasal	37.2	23.5	45.5
sync	36.5	24.1	42.8
classifier	79.3	66.2	65.0
posclass	**81.4**	**71.0**	**66.5**
mturk	100.0	100.0	–n/a–
oracleturk	100.0	100.0	–n/a–

Table 3: Evaluation results. The first column measures how often the channels are selected correctly for both trigger and action (e.g. Android_Phone_Call and Google_Drive in Figure 1). The next column measures how often *both* the channel *and* function are correctly selected for both trigger and action (e.g. Android_Phone_Call::Any_phone_call_missed and Google_Drive::Add_row_to_spreadsheet). The last column shows balanced F-measure against the gold tree over all productions in the proposed derivation, from the root production down to the lowest parameter. We show results on (a) the full test data; (b) omitting descriptions marked as non-English by a majority of the crowdsourced workers; (c) omitting descriptions marked as either non-English or unintelligible by the crowd; and (d) only recipes where at least three of five workers agreed with the gold standard.

	INPUT	Park in garage when snow tomorrow	
(a)	IFTTT	Weather : Tomorrow's_forecast_calls_for	⟹ SMS : Send_me_an_SMS
	OUTPUT	Weather : Tomorrow's_forecast_calls_for	⟹ SMS : Send_me_an_SMS
	INPUT	Suas fotos do instagr.am salvas no dropbox	
(b)	IFTTT	Instagram : Any_new_photo_by_you	⟹ Dropbox : Add_file_from_URL
	OUTPUT	Instagram : Any_new_photo_by_you	⟹ Dropbox : Add_file_from_URL
	INPUT	Foursquare check-in archive	
(c)	IFTTT	Foursquare : Any_new_check-in	⟹ Evernote : Create_a_note
	OUTPUT	Foursquare : Any_new_check-in	⟹ Google_Drive : Add_row_to_spreadsheet
	INPUT	if i post something on blogger it will post it to wordpress	
(d)	IFTTT	Blogger : Any_new_post	⟹ WordPress : Create_a_post
	OUTPUT	Feed : New_feed_item	⟹ Blogger : Create_a_post
	INPUT	Endless loop!	
(e)	IFTTT	Gmail : New_email_in_inbox_from	⟹ Gmail : Send_an_email
	OUTPUT	SMS : Send_IFTTT_any_SMS	⟹ Philips_hue : Turn_on_color_loop

Table 4: Example output from the posclass system. For each input instance, we show the original query, the recipe originally authored through IFTTT, and our system output. Instance (a) demonstrates a case where the correct program is produced even though the input is rather tricky. Even the Portuguese query of (b) is correctly predicted, though keywords help here. In instance (c), the query is underspecified, and the system predicts that archiving should be done in Google Drive rather than evernote. Instance (d) shows how we sometimes confuse the trigger and action. Certain queries, such as (e), would require very deep inference: the IFTTT recipe sets up an endless email loop, where our system assembles a strange interpretation based on keyword match.

instance, more effectively break such ties by learning a prior over which channels are more likely. Turkers, on the other hand, have neither specific training at this job nor a background corpus and more frequently disagree with the gold standard. Second, there are a number of non-English and unintelligible descriptions. Although the turkers were asked to skip these sentences, the machine-learning systems may still correctly predict the channel and action, since the training set also contains non-English and cryptic descriptions. For the cases where humans agree with each other and with the gold standard, the best automated system (posclass) does fairly well, getting 81% channel and 71% function accuracy.

Table 4 has some sample outputs from the posclass system, showing both examples where the system is effective and where it struggles to find the intended interpretation.

6 Conclusions

The primary goal of this paper is to highlight a new application and dataset for semantic parsing. Although if-this-then-that recipes have a limited structure, many potential recipes are possible. This is a small step toward broad program synthesis from natural language, but is driven by real user data for modern hi-tech applications. To encourage further exploration, we are releasing the URLs of recipes along with turker annotations at http://research.microsoft.com/lang2code/.

The best performing results came from a loosely synchronous approach. We believe this is a very promising direction: most work inspired by parsing or machine translation has assumed a strong connection between the description and the operable semantic representation. In practical situations, however, many elements of the semantic representation may only be implied by the description, rather than explicitly stated. As we tackle domains with greater complexity, identifying implied but necessary information will be even more important.

Underspecified descriptions open up new interface possibilities as well. This paper considered only single-turn interactions, where the user describes a request and the system responds with an interpretation. An important next step would be to engage the user in an interactive dialogue to confirm and refine the user's intent and develop a fully-functional correct program.

Acknowledgments

The authors would like to thank William Dolan and the anonymous reviewers for their helpful advice and suggestions.

References

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52, Sofia, Bulgaria, August. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-13)*.

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, Los Angeles, California, June. Association for Computational Linguistics.

S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP)*, Singapore.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 263–270, Ann Arbor, MI.

J. Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37 – 46.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL)*, pages 961–968, Sydney, Australia, July.

Sumit Gulwani and Mark Marron. 2014. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *SIGMOD*.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 913–920, Sydney, Australia, July. Association for Computational Linguistics.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1062–1068, Pittsburgh, PA, July.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session*, Prague, Czech Republic, June.

Klaus Krippendorff. 1980. *Content Analysis: an Introduction to its Methodology*. Sage Publications, Beverly Hills, CA.

Vu Le, Sumit Gulwani, and Zhendong Su. 2013. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *MobiSys*.

Greg Little and Robert C. Miller. 2007. Keyword programming in java. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 84–93, New York, NY, USA. ACM.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Franz Josef Och, Christoph Tillmann, and Hermann Ney. 1999. Improved alignment models for statistical machine translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28, University of Maryland, College Park, MD, June.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA, June. Association for Computational Linguistics.

Yuk Wah Wong and Raymond J. Mooney. 2007a. Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 172–179, Rochester, NY.

Yuk Wah Wong and Raymond J. Mooney. 2007b. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 960–967, Prague, Czech Republic, June.

William A. Woods. 1977. Lunar rocks in natural English: Explorations in natural language question answering. In Antonio Zampoli, editor, *Linguistic Structures Processing*. Elsevier North-Holland, New York.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1050–1055, Portland, OR, August.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *In Proceedings of the 21st Conference on Uncertainty in AI*, pages 658–666.