SpecNFS: A Challenge Dataset Towards Extracting Formal Models from Natural Language Specifications

Anonymous ACL-IJCNLP submission

Abstract

Can NLP assist in building formal models for verifying complex systems? We study this difficult challenge in the context of parsing Network File System (NFS) specifications. We define a semantic dependency problem over SpecIR, a representation language we introduce to model sentences appearing in NFS specification documents (RFCs) as IF-THEN statements, and present an annotated dataset of 1,198 sentences. We develop and evaluate semantic dependency parsing systems for this problem. Evaluations show that even when using a state-of-the-art language model, there is significant room for improvement, with the best models achieving an F1 score of only 60.5 and 33.3 in the named-entity-recognition and dependency-link-prediction sub-tasks, respectively. We also release additional unlabeled data and other domain-related texts. Experiments show that these additional resources increase the F1 measure when used for simple domain-adaption and transfer-learning–based approaches, suggesting fruitful directions for further research.

1 Introduction

Complex software systems are often designed and implemented based on well-defined specifications. A first step towards the formal verification of such a system is to convert its specification, which is often specified in natural language, into a formal model (Soeken et al., 2014). This conversion process is often laborious and error-prone, and a major hindrance to systematic verification of complex systems (Drechsler et al., 2012; Soeken et al., 2014). We ask here if Natural Language Processing (NLP) techniques can assist with automating the process of building formal semantic representations of specification texts. We introduce an instance of this problem in the context of verifying Network File System (NFS) specifications.

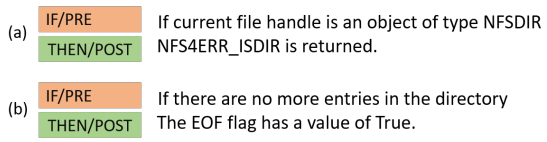


Figure 1: Example specifications from an NFS RFC text seen as pre- and post-conditions.

NFS (Shepler et al., 2008) is a widely used protocol that provides access to files across local and wide-area networks. Implementations of this protocol are expected to meet certain specifications that are laid out in detail in what are called reference RFC (Request for Comment) documents, published by the IETF. These lengthy RFCs (over 500 pages long) specify in natural language how each NFS operation must behave under certain inputs and conditions. Verifying an NFS implementation involves building a semantic representation of these specifications, which can then be used to construct formal models. As with any complex system, NFS protocols, implementations, and specifications undergo multiple draft iterations during a multi-year design process, which includes prototype development and regular interoperability testing. Providing tools that can assist with the construction of the formal models can greatly speed up the development and verification of these complex systems.

Our contributions. We formulate a semantic dependency parsing problem over sentences in NFS RFC documents. We first introduce an intermediate semantic representation language, SpecIR, which captures the specifications for NFS operations (e.g., READ, WRITE) as IF-THEN statements. The IF-part asserts certain pre-conditions, typically defined over the variables involved in the operations, which when satisfied should lead to a post-condition captured by the THEN-part, as illustrated in Figure 1. The pre- and post-conditions are expressed using predicates, functions, their conjunction-disjunction

and basic logical operators. To assist with the development of semantic parsing systems, we introduce, SpecNFS, a dataset of natural language specification sentences annotated with their semantic representations. We formulate two tasks over this dataset [1]. The first is a "sequence tagging" task that involves identifying spans of text that correspond to the main elements in the semantic representation. The second is a "semantic dependency parsing" problem that involves identifying named dependencies between these spans in the sentence.

SpecNFS poses multiple difficulties that are a reflection of the challenges that underlie the application setting of building formal models from specifications. The texts are heavily domain-specific as they talk about components of a complex network software system. Also, given the difficulty of annotation and the overall amount of text available, the amount of labeled data will be relatively limited. To help address these challenges, SpecNFS also includes a collection of unlabeled, broadly related texts that can be used for domain adaption and transfer-learning strategies.

To benchmark the challenges of this dataset, we first evaluate the performance of sequence tagging when fine-tuning large pre-trained language models. Then, for semantic dependency parsing, we evaluate a neural arc-factored model (Dozat and Manning, 2018, 2017), and a neural transition-based parser (Fernández-González and Gómez-Rodríguez, 2020). Evaluations show that (1) there is significant room for improvement in both tasks, and that (2) transfer-learning and simple domain (and task) adaptive pre-training strategies (Gururangan et al., 2020) show significant improvements. Error analyses reveal multiple difficulties arising from entity boundary errors, pipeline errors, and long-term dependencies.

2 Formal Models as Semantic Parsing

Formal modeling refers to the process of stating the expected behavior of a system in a precise formal language. The text of an NFS RFC specifies the expected behavior of NFS operations as constraints defined on input/output-level behavior, system state, and on other important intermediate steps. We can view the process of translating the specifications in text to formal statements as a form of semantic parsing. There are a multitude of formal verification systems such as SysML (Friedenthal et al., 2014), UML (Rumbaugh et al., 1999), SPIN (Holzmann, 1997) etc., each with their own formal language. Rather than use a specific formal language, we introduce SpecIR, a system-agnostic intermediate representation. Figure 2 shows the overview of a system we envision for converting logical specifications in natural language to a system-executable form. A semantic dependency parser converts text into a structured SpecIR representation, which is then converted into a target formal language by a system-specific syntactic parser.

2.1 Data Source Description

We discuss how we tackle the problem of converting specifications expressed in textual form to a structured representation given in SpecIR. In particular, we focus on the core operations expected of an NFS implementation. An NFS RFC provides exhaustive definitions and descriptions of various concepts pertaining to the implementation, operation, and use of NFS. More specifically, the sentences in the *description* and *Implementation* section of NFS operations contain the logical constraints and the recommended strategies for the correct implementation. For example, consider the following sentence from the description section of the READ operation:

> In the case that the current filehandle represents an object of type NFS4DIR, NFS4ERR_ISDIR is returned.

This statement specifies an expected behavior through an implication, an IF-THEN statement, where the IF and THEN conditions themselves are asserted via conditions on the values of variables. The IF-part checks if the **variable cfh** takes on the value **NFS4DIR**. The phrase "current filehandle" in the text refers to the variable **cfh**. If the condition is satisfied, then the operation (whose description is being considered) should return the **value NFS4ERR_ISDIR**. This constraint can be expressed through a dependency graph as shown in the output of the semantic parser in Figure 2.

2.2 Representing Specifications with SpecIR

We introduce SpecIR, an intermediate representation for specifications, as a step towards expressing the underlying logical meaning of specification sentences. The elements of this representation and the annotation scheme we describe next are

[1] Task dataset and description are included in the supplementary material

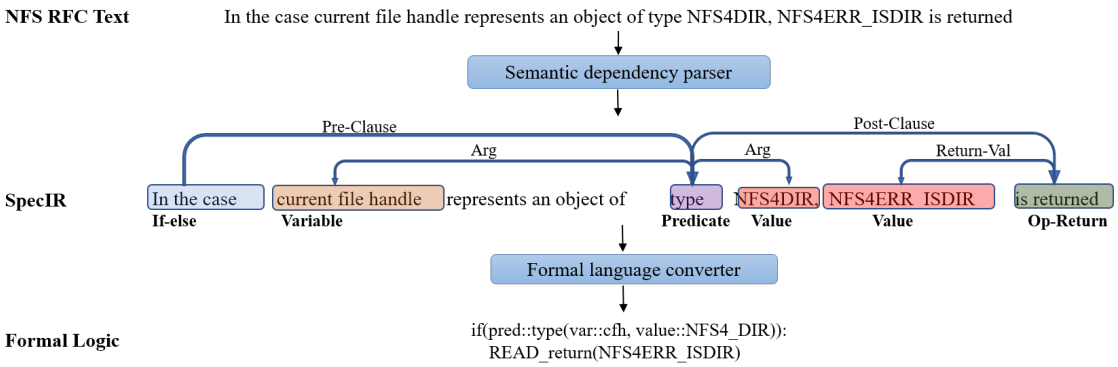


Figure 2: Overview of process for converting a logical statement in natural language to a system-executable form. Sentences are first converted into a system-agnostic logical representation using a semantic dependency parser. Structured output is then converted into an executable form using a formal language converter.

Entity Types	**Coarse Description**
Value	Numbers, Constants etc
Variable	Place-holder for values
Op-name	NFS operations as objects
Function	Assert event or attribute getter
Predicate	Assert property about an object
Op-Return	Return value of NFS operation
And/Or	Logical operations
If-else	Implication indicator
Link Types	**Coarse Description**
Pre-Clause	Points to pre-condition
Post-Clause	Points pre- to post-condition
Arg	Points to arguments of functions, predicates etc.
And-Or-Arg	Points to arguments of And/Or.
Return-Val	Points to the value returned by functions, NFS operations etc.

Table 1: Types of entities and links in SpecIR.

designed based on inputs from domain experts (researchers who focus on NFS). As mentioned earlier, the specifications assert constraints over elements in the NFS RFC (e.g., variables, constants, method names). Accordingly, SpecIR includes NFS elements, uses logical operators (And, Or) to connect constraints, and uses an IF-ELSE construct to convey the overall specification. SpecIR in essence is a dependency structure that can be layered over a given specification sentence. To scope the semantic dependency problem, we ignore specifications that require connecting information from multiple sentences. As can be seen from the example in Figure 2, representing a specification in SpecIR involves labeling spans with entities and links:
(i) **Entities**: Words or phrases in a sentence that belong to specific categories that indicate their semantic type and role in the overall semantics of the sentence.
(ii) **Links**: Labeled directed edges that convey how the individual entities compose to express the overall logical semantics of the sentence.

3 SpecNFS Dataset

The annotation process had three phases: initial annotation, review, and disagreement resolution.

Initial Annotation. We annotated about 1,600 sentences that express specifications in the IF-THEN format described earlier. We discard cases where the specification is spread over multiple sentences, and ones that require complex temporal logic that are outside the scope of SpecIR. These sentences were collected from the operations description in NFS RFC 4.0, 4.1 and 4.2. Five annotators and two domain experts were involved in the annotation process. Of the two experts, one is a Professor who has more than 30 years of extensive research experience in network file systems and operating systems. The experts designed the annotation scheme, reviewed the annotations, and resolved disagreements among annotators. Each annotator was given around 320 sentences to annotate, a detailed set of guidelines (Appendix 7.1) , and an initial round of training.

Review. The annotations were jointly reviewed by the experts and the annotators after 200 sentences were annotated. Each annotator reviewed 10 sentences annotated by another annotator. Any disagreement between the reviewer and the original

Type	#Analyzed	#Edits	Percentage
Entities	497	20	4%
Links	460	25	5.4%

Table 2: Annotation quality in terms of edits required by an expert for correcting discrepancies.

annotator was delegated to an expert to be resolved. Also, during this review, annotators collected sentences that were difficult to precisely annotate under the current annotation scheme.

Disagreement resolution. An expert decided on the correct annotation when a disagreement arose between the annotator and the reviewer. If the disagreement signaled a systematic annotation error by one of the annotators, the expert conducted a further review of a random set of sentences of that annotator to check for the presence of such a systematic error. In cases where the annotation scheme was the cause of the disagreement, the scheme was updated and the previous annotations were reviewed and adjusted to reflect the new changes.

Annotation quality. After multiple iterations, we ended up a with collection of 1,198 annotated sentences having 9,358 entities and 6,872 links. Due to the iterative nature and complexity of the annotation process, rather than measure the inter-annotator agreement, here we focus on the overall annotation quality of the dataset, as determined by the expert annotators. We collected 50 annotated sentences, 10 from each annotator, and had an expert analyze them. For each sentence, the expert judged whether they agree with the annotation and computed the minimum edits in the annotation required to fix discrepancies if any.

Table 2 shows the results of the expert's review of the annotation quality. The overall low numbers of edits required to fix the annotation discrepancies indicates low label noise in the annotation. The main source of label noise for the entity labeling is identification of the correct span boundary and the ambiguity in the usage of **Predicate** and **Function** labels. For the label noise for the dependency links, this is mainly due to the sparse existance of links between any two entities in a sentence.

Unlabeled Domain Relevant Texts. Interpreting specification texts requires reasoning about concepts in the target system domain (i.e. NFS), in addition to tackling the linguistic aspects and other usual difficulties that arise in producing semantic representations (e.g. annotation accuracy, inducing consistency). This is not unique to the NFS systems that we target and highlights the difficulty in obtaining large-scale annotated data for this type of problem. One way to address this is to leverage additional domain-relevant unlabeled data. To this end, we collected texts broadly related to the NFS domain shown in Table 3. As we show later, these unlabeled texts help when used for domain-adaptive transfer (Gururangan et al., 2020).

Source	**Description**	**#Tokens**
RFCs	Internet standards	23.2m
Man pages	Docs for other NFS-like file systems	5.5m
Research	Articles on file systems	4.7m
SRS	Project requirements	1.1m
Github	Open source code	129.5m

Table 3: Domain-relevant texts used to fine-tune the models for downstream NFS tasks.

4 Semantic Dependency Parsing

We formulate the task of converting textual specifications into statements in SpecIR as a semantic dependency parsing problem with two steps. The first step identifies the spans of entities in the text, and the second identifies semantic dependency links between these entity spans. We first benchmark standard approaches for each step (subsection 4.1, subsection 4.2) and then present error analyses that reveal their respective challenges (subsection 4.3).

4.1 Named Entity Recognition

This is a standard sequence tagging task, where the task is to identify spans referring to the various entities in our annotation scheme listed in Table 1. We benchmark sequence tagging solutions using four large language models: BERT (Devlin et al., 2019), DistilBERT (Sanh et al., 2020), CS RoBERTa (Gururangan et al., 2020) trained on articles from CS Research, and CodeBERT trained on github data(Feng et al., 2020).

We also explore domain adaptation strategies that can utilize other unlabeled domain-relevant texts. Even though BERT-like models generalize fairly well across many domains (Beltagy et al., 2019), (Lee et al., 2020) have shown that pre-training on such domain-specific corpora boosts the model performance on downstream tasks. Ta-

Model	base	NFS-DAPT	NFS-TAPT
BERT	59.5±0.3	59.5±0.3	59.5±0.3
DistilBERT	58.6±0.3	58.9±0.3	59.4±0.3
CodeBERT	59.1±0.3	59.3±0.4	60.1±0.4
CS RoBERTa	59.8±0.5	60.2±0.3	**60.5**±0.3

Table 4: Mean and std. deviation macro F1 scores for named entity recognition across 5-Folds CV.

ble 3 gives a brief overview of the various sources (Miceli Barone and Sennrich, 2017), (Gelman et al., 2019), (Ferrari et al., 2017) we used to further fine-tune the four base models on the *Masked LM* task as in (Devlin et al., 2019). We follow the domain-adaptive (**DAPT**) and task-adaptive (**TAPT**) formulations described in (Gururangan et al., 2020). For the experiments with DAPT, our goal is to expose the model to a broad collection of texts related to system specifications, NFS RFCs, and code elements. To this end, we fine-tune on a collective corpus of ∼850 MB. For TAPT, the goal is to focus on a narrower collection of task domain text. We only use the NFS RFCs, a ∼2 MB subset of the RFC dataset, as they are highly representative of the target task data.

4.1.1 Benchmarking Results

We fine-tune and evaluate the models from Section 4.1 on the NER task for the entities specified in Table 1. We treat the task as a multi-class classification problem, following the **B-I-O** labeling scheme (Ramshaw and Marcus, 1995) at the word-level. For a sentence with n tokens $w_1, \ldots, w_n$, the computation involves producing a contextual token representation x_i for each token w_i which is then passed through a softmax classifier to predict a **B-I-O** token label sequence $c_1, \ldots, c_n$. Note that the tokenizer might break a word into sub-tokens; so only the head token of a word is tagged, while the rest are labeled as "[X]" and ignored while calculating loss. During inference, we derive the label of a word based only on its head token.

Given the small size of the dataset, we evaluate the models using stratified five-fold cross-validation repeated for five different initialization.

Table 4 shows the results for all models in terms of word-level F1 scores obtained as a macro-average over all entity types. Similar to the results in (Gururangan et al., 2020), combined DAPT and TAPT fine-tuning (referred to as NFS-TAPT) performs better than both the base and DAPT-only (referred to as NFS-DAPT) fine-tuning. While some entity types are easier to recognize (e.g. Value), there are multiple types that are harder (e.g. Function, Predicates), resulting in overall low F1 scores. Overall while large language models show promise, this NER task presents a difficult challenge with a clear room for improvement.

4.2 Dependency Link Prediction

Predicting the dependency link between entities can be framed as a dependency parsing task defined over multi-word spans. Consider a sentence with n tokens $S = \{w_1, \cdots, w_n\}$ and k entities $e_1, ..., e_k$, where each entity corresponds to a contiguous sequence of one or more tokens. The task, then, is to predict the dependency link $l_{ij} \in L$ between the entity pairs e_i and e_j $\forall i, j \in \{1, ..., k\}$, where L is the set of possible link types. Note that the task is defined only over entity pairs: words that are not part of entities are not considered for link prediction. We evaluate two complementary approaches: arc-factored parsing and transition-based parsing.

4.2.1 Arc-Factored Parsing

We benchmark a variant of the arc-factored model described in Dozat and Manning (2018) and Dozat and Manning (2017). The core idea is to get *head* and *dependent* representations from the contextual representations of each word using two separate feed-forward transformations (MLP layers). The head and dependent representations of an entity $e_j = w_l, ..., w_{l+p}$ in a sentence S is given by :

$$
\begin{aligned}
x_i &= \mathbf{LM}(w_i; S) \\
e_j^W &= \mathbf{max}(x_l, \ldots, x_{l+p}) \\
u_j &= e_j^W \oplus e_j^T \\
u_j^{dep} &= \mathbf{MLP}^{(dep)}(u_j),\ u_j^{head} = \mathbf{MLP}^{(head)}(u_j)
\end{aligned}
$$

Here, **LM** is a standard pre-trained language model (e.g. BERT), e_j^W is the contextual representation of e_j using the **LM** and e_j^T is the entity type emedding of e_j. The probabilities of the dependency link labels from entity e_r to e_j is computed as follows:

$$
\begin{aligned}
u_{rs} &= u_r^{head} \oplus u_s^{dep} \\
q_{rs} &= (u_r^{head})^T W_1 (u_s^{dep}) + W_2 u_{rs} + b \\
y_{rs} &= \mathbf{softmax}(q_{rs}) \qquad (1)
\end{aligned}
$$

The parameters of the model are learned by minimizing the cross- entropy between the true and the predicted link label between the entities.

4.2.2 Transition-Based Parsing

For the transition-based parsing method, we adapt the system developed by Fernández-González and Gómez-Rodríguez (2020). This system uses an encoder-decoder model with LSTM Networks. The encoder is a bi-directional LSTM (BiLSTM) which generates the contextual representations of the tokens in the input sentence. The decoder is another LSTM which uses the encoder's outputs and a pointer network (Vinyals et al., 2015) to make sequential linking decisions.

Given an input sentence $S = \{w_1, \ldots, w_n\}$, for each token w_i, we obtain a token-level embedding (e_i^W) either using BERT-base or using Glove (Pennington et al., 2014), character-level embedding (e_i^C), the embedding for the lemmatized version of the token (e_i^L), and the embedding for its entity type (e_i^T). We concatenate all four embeddings and feed it to a BiLSTM to get contextualized representations. The entities are represented by the encoder hidden states (i.e. the contextual representations) of the first token in their respective spans.

$$e_i^W = \textbf{LM}(w_i; S) \; or \; \textbf{GLOVE}(w_i; S)$$
$$x_i = e_i^W \oplus e_i^T \oplus e_i^C \oplus e_i^L$$
$$h_i = \textbf{BiLSTM}(x_i)$$

The decoder generates a sequence of transition decisions using these hidden states, following the steps in Fernández-González and Gómez-Rodríguez (2020). At time t, attention score a_t is predicted between the current focus word h_i and other tokens in the sentence, using the last predicted head word h_h. This is followed by predicting the link label between the token with the highest attention score, h_p and s_t.

$$r_i = h_i + h_h$$
$$s_t = \textbf{LSTM}(r_i)$$
$$v_j^t = f_1^T(s_t) W f_2(h_j) + U^T f_1(s_t) + V^T f_2(h_j) + b$$
$$a_t = \textbf{softmax}(v^t)$$
$$q_{tp}^l = g_1^T(s_t) W_l g_2(h_p) + U_l^T g_1(s_t) + V_l^T g_2(h_j) + b_l$$
$$y_{tp} = \textbf{softmax}(q_{tp})$$

where $j \in 1, .., n$ and y_{tp}^l is the probability of word w_p being the head of a focus word at time t through link type l. For decoding, we use beam search with a beam size of five. The parameters are learned by maximizing the likelihood of generating the correct sequence of decisions for the corresponding parse tree.

4.2.3 Benchmarking Results

To benchmark the effectiveness of above approaches, we use stratified 5-fold cross-validation, repeated five times with different parameter initialization, on the same folds that were used to report the named entity recognition model performance. We compute mean macro F1 scores by averaging the F1 scores of each of the five link types listed in Table 1 and averaging it over the five initializations.

For the arc-factored system, we also assess the benefits of transfer learning and domain adaptation strategies and the utility of adding type constraints during inference. We report results of the following five variants: (i) *LM Base* – the system where pre-trained language model (*LM*) is initialized with it's standard pre-trained weights. (ii) *LM Base + NER* – the pre-trained *LM* model that was fine-tuned on the Named Entity Recognition (NER) task. (iii) *LM Base + NER + DAPT* – the NER fine-tuned *LM* with *DAPT*. (iv) *LM Base + NER + TAPT* – the NER fine-tuned *LM* with *TAPT*. (v) *LM Base + NER + TAPT + Link Constraints* – the TAPT version which only considers the link labels that are valid for the types of the entities under consideration. We report results with BERT, CSRoBERTa, DistilBERT and CodeBERT as the *LM* in all these variants. For all variants, the entity type embedding and other non-pre-trained *LM* components of the model are initialized with the Xavier method (Glorot and Bengio, 2010) and trained end-to-end.

For the transition-based system, we report results for two variants that differ in the inputs to the BiLSTM layer: (i) Glove Embeddings – uses 100 dimensional Glove embeddings as input to the BiLSTM. (ii) BERT-base – uses frozen embeddings from pre-trained BERT-base as input to the BiLSTM. Due to the difficulty of replicating standard fine-tuning within this system's implementation, we do not report transfer-learning experiments here. Our initial transfer-learning experiments with frozen embeddings were also unsuccessful. Table 5 shows the results of all the systems. Using task-transfer and adaptive training strategies, we observe substantial increases in F1 over directly using the pre-trained **LMs**. Initializing with the NER fine-tuned weights nearly doubles the F1 compared to the pre-trained weights alone, for all the pre-

Model class	Language Model	Base	Base + NER	Base + NER + DAPT	Base + NER + TAPT	Base + NER + TAPT + Link Constraints
Arc factored	BERT	16.1±1.7	32.0±1.1	31.7 ± 0.7	32.8 ± 2.0	**33.3 ± 2.1**
	DistilBERT	14.2±0.7	15.1±1.2	14.2 ± 0.7	15.5 ± 1.2	15.7 ± 1.2
	CS RoBERTa	6.3 ± 2.0	31.6±1.1	32.8 ± 0.08	32.1 ± 1.0	32.4 ± 0.01
	CodeBERT	18.0±2.1	32.0±0.8	30.8 ± 1.4	31.1 ± 1.3	31.7 ± 1.1

Table 5: Benchmarking results for arc-factored systems: Mean macro F1- scores and standard deviation values computed over five different random initializations.

Link type	Pre	Arg	Ret-val	Post	And-Or-Arg
BERT	18.7	11.2	20.5	1.9	28.1
TAPT+LC	45.5	28.3	37.9	11.9	43.0

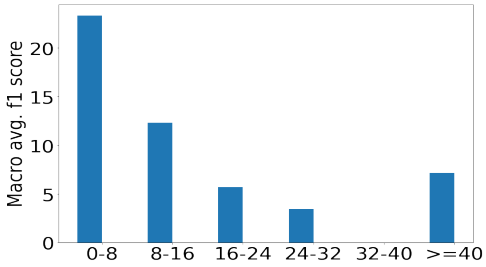
Table 6: Link-wise F1 scores for the Dependency Link Prediction task for BERT-base and transfer-learning with link constraints (TAPT+LC).

Model Class	Language Model	Macro-F1
Transition	BERT	10.3 ± 0.4
	Glove	23.3 ± 1.5

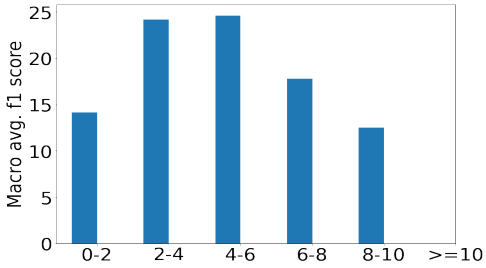
Table 7: Benchmarking results for transition-based systems: Mean macro F1- scores and standard deviations computed over five different random initializations.

trained LMs except for DistilBERT. For BERT and CSRoBERTA based models, adding task-adaptive training (TAPT) results in a ∼0.5 to 1% increase in F1 relative to the NER fine-tuned initialization. Furthermore, enforcing the type constraints on the link prediction during inference yields small additional gains in F1. Table 6 shows the break-down across the different link types. The post link which often requires long-distance tracking, turns to out to be the most difficult. For the transition-based system, as shown in Table 7, using the high dimensional frozen BERT-embeddings fares poorly compared to Glove embeddings as input to the LSTM. While contextual embeddings from large language models tend to outperform static embeddings in general, in our setting there are two key differences that could have caused the unexpected result. First, the BiLSTM is able to provide some contextual information even with Glove embeddings, and second training with large frozen embeddings on relatively smaller dataset such as ours can be tricky.

Note that the dependency link prediction task we



(a) F1-score vs link terminal's span length.



(b) F1 vs average no. tokens in the two terminal entities of a link

Figure 3: Figure 3(a) shows that as the span of a link's terminals increases, the macro-average F1 score for the link type prediction decreases. Figure 3(b) shows issues with long-term dependencies. Macro-average F1 scores drop as the distance between the head and the dependent terminal of a link increases.

evaluated here assumes that the gold label for the entity spans are given as input. The results here should be seen as an upper bound for performance of end-to-end systems that produce the dependency parse from the input sentence annotated with the automatically labeled named entities.

4.3 Error Analysis

Named Entity Recognition We analyzed 1,471 mis-classified samples from the BERT-NFS-TAPT model. The analysis reveals two main sources of errors:**(i) Functions vs. Variables:** About 19% of the Functions are predicted as Variables. This error

stem from the fact that often function of a variable can be a valid variable. For example in the phrase *size of the file*, we can interpret $size$ as a $function$ operating on the $variable\ file$ but then , the entire phrase can be interpreted as a valid $variable$.**(ii) Data imbalance:** A closer inspection of the results show that the model is much better at predicting the beginning of a entity with a macro F1 score of 65%, while the F1 score for predicting the inside of a entity drops to 24%. In fact, for the test set, the model does not predict any I-{Op-Return, Function, and Value} tags, indicating the data imbalance issues in modeling multi-token entities.

Link Prediction We analyzed a random sample of the 1,183 links predicted by the arc-factored model. We found two main categories of errors: **(i) Span information smoothing**: For 45% of the links that are mis-classified, either the head or the dependent of the link spans more than two tokens. Figure 3(a) shows that errors increase with the span length of the entities involved, suggesting the need for a more effective means for aggregating embeddings of the tokens spanned by an entity. **(ii) Long-term dependency**: About 40% of the mis-classified links have their head and dependent terminals separated by more than 8 tokens. Figure 3(b) shows that as the distance between the head and the dependent terminals of the link increases, the model performance decreases.

5 Related Work

We discuss two types of related work:

(i) NLP for formal verification. Drechsler et al. (2012) use syntactic analysis to convert textual specifications into *UML* or *OCL* descriptions. Pandita et al. (2012) use pre-defined templates over syntactic structures to extract semantic representations of API documents. Soeken et al. (2014) use an additional clustering step to seed the process for manually crafting templates. Harris and Harris (2016) uses custom formal grammar instead of manually crafted templates to capture sentence structure of specifications. Such manually-engineered templates or grammars are suitable for settings with high-level of regularity in the manner in which specifications are expressed in natural language. In this work, we seek to further automate the process and ask if we can train parsing models to learn from example semantic parse annotations. To this end, we release a challenge dataset and benchmark the performance of strong neural parsing baselines for this task.

(ii) NLP for analyzing RFCs. (Landhosectionuoer et al., 2012) demonstrate the transfer of modifications between the specification text and corresponding UML models to avoid inconsistencies. (Jero et al., 2019) leverage network protocol RFCs to extract relevant packet fields and their properties and use them in grammar-based fuzzing (Jero et al., 2015) to uncover system vulnerabilities. (Tahir and Oswald, 2012) and (Yen et al., 2020) propose systems to convert specifications into logical representations and eventually code snippets. While the former simply trains a Penn Treebank (Marcus et al., 1993) parser to represent the text, the latter uses CCG (Artzi et al., 2014) parsing combined with a domain-specific lexicon to generate the semantic representations. Both works deal with relatively smaller texts, with 22 and 87 sentences, respectively. With about 1,000 sentences across more than 40 different NFS operations, the source text for SpecIR is more varied and the resultant model is likely to generalize better.

6 Conclusions

In this paper, we introduced the problem of extracting formal models from Network File System RFCs and took some significant steps towards addressing this problem. We designed an intermediate representation, SpecIR, that expresses a specification in terms of IF-THEN statements, and we introduced a challenge dataset, SpecNFS, that contains SpecIR annotations of NFS RFC sentences. Benchmarking experiments show that this is a challenging dataset for the semantic dependency parsing models we explored. The improvements we see with basic domain and task-adaptive methods show promise for further research into transfer-learning strategies.

Semantic dependency parsing on RFC documents also presents unique opportunities to test recent advances in NLP, motivating future research. One direction lies in exploiting the rich structure of the NFS RFC using structure-specific pre-training methods for tables (Yin et al., 2020), and code elements (Feng et al., 2020). The difficulty of annotation—and the cost involved in the process—also warrants exploring human-in-the-loop processes through active, interactive learning, zero and few-shot generalization methods. We hope that the dataset and benchmarks we release will spur further research along these directions.

7 Appendices

7.1 Annotation Scheme

This section gives a brief overview of the heuristics followed by the annotators to annotate the sentences with their SpecIR representation.

SpecIR representation of a textual specification is based on the concepts of first-predicate logic, which involves predicates and functions, applied to appropriate objects in order express some proposition. For every sentence in the NFS Operations specifications, the annotators tried to label *entities* and then the *link* between them. Few examples of the dependency parse of the logical specification, generated following our annotation scheme can be seen in Figure 4.

7.1.1 Entities

These are tokens or sequences thereof that represent potential elements of SpecIR. The SpecIR representation has the following elements:

1. Objects

1.1 Value: Raw values or constants assigned to variables or returned from a function. Raw values include numbers and Boolean values *True* and *False*. Constants include all the NFS4_*X* or NFS4ERR_*X* states that indicate the success or failure of an operation.

1.2 Variable: These are similar to the variables used in a program. While annotating a particular operation, variables could either be part of the argument and return sections of the operation, or could be generic placeholders for the values.

1.3 Op-name: Since these are basically NFS4 operations, any mentions of the NFS4 operations in a sentence were tagged as such as long as only the name was mentioned. If the sentence also mentioned its behavior, Function tag was used.

2. Function: These either denote an action performed during the run of the code or refer to some attribute of an object. Functions can have values, variables, and return states of other functions as arguments—and can return another variable or value. The semantics of the word or phrase in a sentence that is tagged as **Function** is indicative of the nature of the function.

3. Op-Return: Similar to a function, describes the behavior of a native NFS4 operation and its return states. If the a sentence has a mention of a function returning something, without explicitly mentioning the function name, then **Op-Return** has to be used.

4. Predicates: Predicates are used to affirm or deny some property about a objects, variables, or events. To decide if a **Predicate** tag is to be used, the annotator answers a Yes/No question about a value or variable. Example cases where predicates should be used are comparison of two or more values/variables to satisfy a pre-condition for an event, confirming a specific state of an object such as a file is empty or EOF has been reached, etc.

5. Connectives

5.1 And/Or: Used for conjunction or disjunction of multiple logical expression or objects. Functions, predicates and objects are the usual arguments of And/Or connective. All the arguments of And/Or connective must be of same type.

5.2 If-else: If-then is an implication connective, which is used to specify implication relation between two logical expression (i.e. one logical expression is a logical consequence of another). The arguments of if-then (i.e. both the premise as well the conclusion) must be a truth value. Based on the truth value of the arguments *premise and conclusion*, If-then evaluates to True or False.

7.1.2 Links

These denote the dependency relationship between annotated entities, if any.

1. Pre-Clause: Links an If-else entity, which indicates an implication, with its relevant precedent. Multiple If-Clause links can be found in sufficiently complex expressions where multiple steps are presented in the same sentence.

2. Argument: Links a **Function, Predicate**, and **Op-Return** entity in a sentence, with their corresponding arguments. The head and the dependent can both have multiple outgoing and incoming **Argument** links, as they can both take arguments as well as act as an argument for other entities.

3. Return-val: Links a function with its return state. The return state could be a value like 0 or an indication of failure such as NFS4_ERR.

4. Post-Clause: Links a pre-condition to its corresponding post-condition event. The event that it links can only be of type **Function, Predicate, Op-return** and their **And/Or** conjunction.

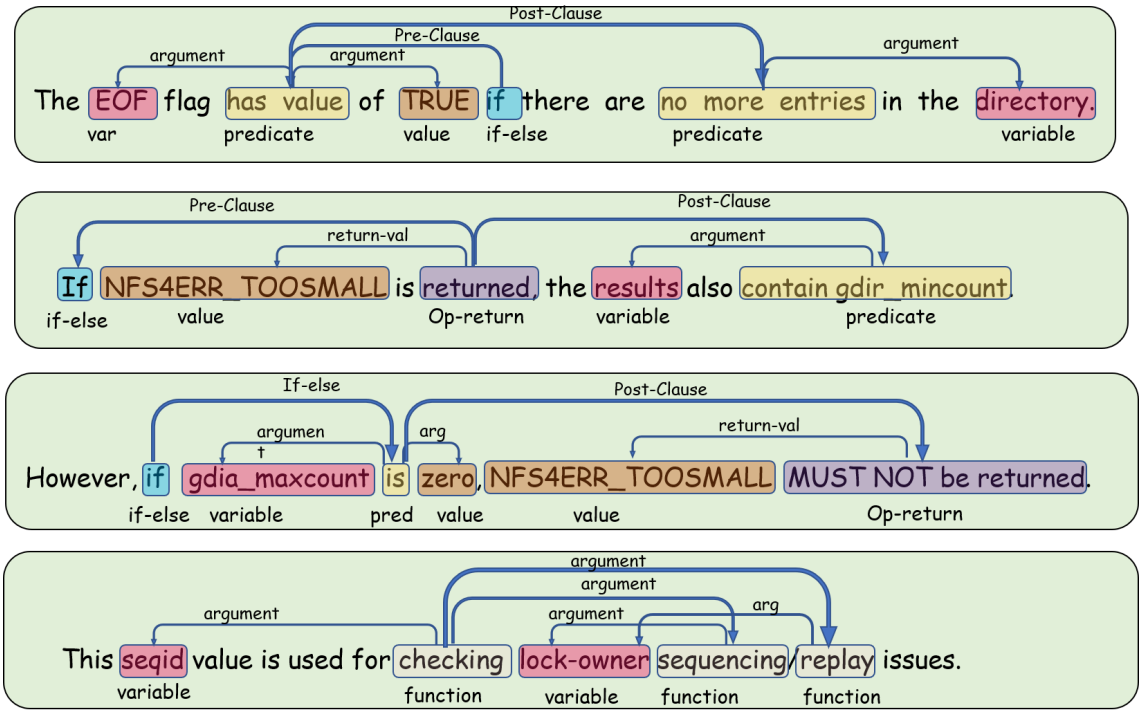


Figure 4: Examples of dependency parse of logical specifications following our annotation scheme.

5. And-Or-Arg: Links a **And/Or** entity to the arguments that it conjoins or dis-joins.

References

Yoav Artzi, Nicholas Fitzgerald, and Luke Zettlemoyer. 2014. Semantic parsing with Combinatory Categorial Grammars. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, Doha, Qatar. Association for Computational Linguistics.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. 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 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. *ArXiv*, abs/1611.01734.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Rolf Drechsler, Mathias Soeken, and Robert Wille. 2012. Formal specification level: Towards verification-driven design based on natural language processing. In *Proceeding of the 2012 Forum on Specification and Design Languages*, pages 53–58. IEEE.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Transition-based semantic dependency parsing with pointer networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7035–7046, Online. Association for Computational Linguistics.

A. Ferrari, G. O. Spagnolo, and S. Gnesi. 2017. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505.

Sanford Friedenthal, Alan Moore, and Rick Steiner. 2014. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann.

Ben Gelman, Banjo Obayomi, Jessica Moore, and David Slater. 2019. Code and comments dataset.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Christopher B Harris and Ian G Harris. 2016. Glast: Learning formal grammars to translate natural language specifications into hardware assertions. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 966–971. IEEE.

Gerard J. Holzmann. 1997. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295.

S. Jero, H. Lee, and C. Nita-Rotaru. 2015. Leveraging state information for automated attack discovery in transport protocol implementations. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12.

Samuel Jero, Maria Leonor Pacheco, Dan Goldwasser, and Cristina Nita-Rotaru. 2019. Leveraging textual specifications for grammar-based fuzzing of network protocols. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:9478—9483.

M. Landhosectionuoer, S. J. Kopgfmer, and W. F. Tichy. 2012. Synchronizing domain models with natural language specifications. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, pages 22–26.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313—330.

A. V. Miceli Barone and R. Sennrich. 2017. A parallel corpus of python functions and documentation strings for automated code documentation and code generation.

Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. 2012. Inferring method specifications from natural language api descriptions. In *2012 34th international conference on software engineering (ICSE)*, pages 815–825. IEEE.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

James Rumbaugh, Ivar Jacobson, and Grady Booch. 1999. The unified modeling language. *Reference manual*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

S. Shepler, M. Eisler, and D. Noveck. 2008. NFS version 4 minor version 1. Technical Report IETF Internet-Draft, Network Working Group.

Mathias Soeken, Christopher B Harris, Nabila Abdessaied, Ian G Harris, and Rolf Drechsler. 2014. Automating the translation of assertions using natural language processing techniques. In *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*, volume 978, pages 1–8. IEEE.

Rashid Tahir and Justin Oswald. 2012. Implementing RFCs using natural language processing.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *NIPS*.

Jane Yen, Tamos Levai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. 2020. Semi-automated protocol disambiguation and code generation.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.