

DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Enhancing Sentence Decomposition in Large Language Models Through Linguistic Features

Author:

Xinyu XU

Student number:

5759846

Thesis Advisor:

Prof. Dr. M.S. PERA

Supervisor:

Prof. Dr. J. YANG

Daily Supervisor:

Gaole HE

*A thesis submitted in fulfillment of the requirements
for the degree of Master of Science*

in the

Web Information Systems Group
Artificial Intelligence

August 12, 2024



Declaration of Authorship

I, Xinyu XU, declare that this thesis titled, “Enhancing Sentence Decomposition in Large Language Models Through Linguistic Features” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Xinyu Xu

Signed:

Date: 2024-08-12

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Artificial Intelligence

Master of Science

Enhancing Sentence Decomposition in Large Language Models Through Linguistic Features

by Xinyu XU

This thesis investigates the enhancement of sentence decomposition in Large Language Models (LLMs) through the integration of linguistic features, including constituency parsing, dependency parsing, and Abstract Meaning Representation. Traditional decomposition methods, which often rely on rule-based approaches, struggle with highly intricate sentences. By incorporating detailed linguistic features and employing reasoning steps and supervision in prompts, we aim to improve the comprehension and decomposition capabilities of LLMs. Experimental results show that integrating these linguistic features has the potential to improve decomposition performance. However, these enhancements also introduce new error types and increase computational costs. This study identifies several error types in the programs generated by LLMs, including format, decomposition, and conversion errors, emphasizing the need for further refinement in model training and prompt design. This research lays a step toward more accurate and efficient processing of complex sentences in LLMs, encouraging ongoing development and optimization in this field.

Acknowledgements

I would like to express my deepest gratitude to my advisor Prof. Dr. M.S.Pera and my supervisor Prof. Dr. J.Yang, whose guidance have been invaluable during the high-pressure period of my thesis project. Your support not only made the intense workload more manageable but also significantly alleviated my anxiety, allowing me to feel better and more confident.

I am also profoundly thankful to my daily supervisor Gaole He. Coming from a background with limited knowledge and experience in this field, I greatly appreciate your meticulous and patient guidance. Your dedication and responsibility towards my work transformed me from a novice to someone who could complete this thesis.

My heartfelt thanks go to my friends Yangyu, Karthik and Vitor. During my thesis work, I was also dealing with job hunting and moving, which were highly stressful endeavors. Your companionship and support helped to ease my anxiety and tension, providing emotional relief during these challenging times. I would like to extend a special thanks to Karthik, who not only provided me with numerous resources when I was at a loss in my research but also reviewed my thesis and offered valuable guidance.

I am deeply grateful to my family, especially my mother, for their nurturing and unconditional support in my study. Your encouragement has allowed me to successfully complete my postgraduate studies.

Lastly, I would like to acknowledge myself. Despite numerous setbacks and challenges during the research process, I managed to graduate on time. The perseverance and effort it took are commendable. Thank you for not giving up and seeing this journey through to the end.

Contents

Declaration of Authorship	iii
Abstract	v
Acknowledgements	vii
1 Introduction	1
1.1 Research question	3
1.2 Contribution	3
1.3 Overview	4
2 Preliminaries	5
2.1 Sentence Complexity	5
2.2 ProgramFC	6
2.2.1 Principle	6
2.2.2 Program Structure and Syntax	6
2.3 Linguistic Features	7
2.3.1 Syntactic Parsing	7
Constituency Parsing	8
Dependency Parsing	8
2.3.2 Semantic Parsing	9
Abstract Meaning Representation	10
2.4 Large Language Models	11
3 Related Works	13
3.1 Sentence Decomposition	13
3.1.1 Intergrating Linguistic Feature	13
3.1.2 Model Enhancement	14
3.2 Prompt Engineering	14
3.2.1 In-context Learning	14
3.2.2 Chain-of-thought Reasoning	16
3.2.3 Problem Decomposition	17
3.3 LLM Enhancement for Decomposition	18
3.3.1 Planning and Decision Making	18
Tree search	18
External Module Intergration	18
3.3.2 Feedback and Self-reflection	18
4 Approach	21
4.1 Motivation and Hypotheses	21
4.2 Overall Architecture	21
4.3 Program Generation	23
4.3.1 Incorporating Linguistic Features	23

4.3.2	Reasoning Methods	25
	No Reasoning	25
	Chain-of-Thought Reasoning	26
	Tree-of-Thought Reasoning	27
4.3.3	Supervision to Reasoning	28
	No Supervision	28
	Supervision	28
4.4	Program Execution	30
4.4.1	Function Handler	30
5	Experiments	31
5.1	Experimental Setup	31
5.1.1	Dataset	31
5.1.2	Models	32
	GPT	32
	Mixtral	33
	Qwen	33
5.1.3	Baseline	33
5.1.4	Study Design	34
5.2	Study One: Impact of Linguistic Features	35
5.2.1	Main Results	35
5.2.2	Observations	36
5.3	Study Two: Impact of Reasoning and Supervision	36
5.3.1	Main Results	36
5.3.2	Observations	37
5.4	Further Analysis on Study Two	38
5.4.1	Program Analysis	38
	Format Errors	38
	Decomposition Errors	39
	Conversion Error	42
5.4.2	Case Study	44
	Why Baseline Outperforms Other Settings?	44
	What is the Distribution of Different Errors in Programs?	45
	Results and Observations	45
6	Discussion	49
6.1	Answer to Research Questions	49
6.1.1	Can the addition of linguistic features enhance the decomposition capabilities of LLMs?	49
6.1.2	How effective are different prompting methods in aiding LLMs reasoning during the decomposition process?	49
6.2	Limitation	50
6.2.1	Experimental Limitation	50
	Limited Data Scope	50
	Reduced Experimental Setting	51
6.2.2	Technical Limitation	51
	Introducing Extra Errors	51
	Ineffective Use of Parsing Tree Information	51
	Dependency on Model and Data	51
6.3	Further Improvement	51
6.3.1	Experimental Improvements	52

6.3.2	Technical Improvements	52
7	Conclusion	53
	Bibliography	55
A	Full Prompts	63
A.1	Baseline	63
A.2	No Reasoning with No Supervision(NR-NS)	65
A.3	No Reasoning with Supervision (NR-S)	71
A.4	Chain-of-Thought with No Supervision(CoT-NS)	81
A.5	Chain-of-Thought with Supervision(CoT-S)	95
A.6	Tree-of-Thought with No Supervision(ToT-NS)	118
A.7	Tree-of-Thought with Supervision(ToT-S)	123

List of Figures

1.1	An Example of LLMs Failing to Understand a Complex Claim	2
1.2	Constituency Tree	2
2.1	Constituency Parsing Tree of Sentence “The boy wants the girl to believe him.”	9
2.2	Dependency Parsing Tree of Sentence “The boy wants the girl to believe him.”	10
2.3	Abstract Meaning Representation of Sentence “The boy wants the girl to believe him.”	10
4.1	Overview of Our Approach	22
4.2	Prompting Exemplars of Different Prompting Methods	24
4.3	Parsing Tree Examples of sentence “The boy wants the girl to believe him.”	25
5.1	Distribution in Program Generation for ‘Why Baseline Outperforms Other Settings’	45
5.2	Error Distribution in Program Generation for ‘What is the Distribution of Different Errors in Programs?’	46

List of Tables

4.1	Prompting Methods	23
5.1	Pricing of Models (in US Dollars)	32
5.2	Experimental Setting for Studies	34
5.3	Macro-F1 Score for Study One	35
5.4	P Values for the Result of Study One	35
5.5	Cost (in US Dollars) for Study One	35
5.6	Macro-F1 scores for Study Two	36
5.7	P Values of the Result of Study Two	37
5.8	Cost (in US Dollars) for Study Two	37

List of Abbreviations

LLM	Large Language Model
NLP	Natural Language Processing
AMR	Abstract Meaning Representation
ICL	In Context Learning
CoT	Chain of Thought
ToT	Tree of Thought

Chapter 1

Introduction

The decomposition of complex sentences into simpler, more manageable components is one of the crucial tasks in the field of natural language processing (NLP), with widespread applications in information retrieval [1], question answering [2], and automated fact-checking system [3]. Complex sentence decomposition involves breaking down intricate statements into simple subclauses, facilitating better understanding and processing by computational models. This task is essential for improving the verifiability and interpretability of models, allowing humans to better understand how these models process and interpret natural language. By making the model's understanding more transparent, we can ensure that complex sentences are handled in a way that is both clear and accurate.

Many prior studies have focused on the task of decomposing complex sentences. Traditional methods often relied on rule-based approaches, dependency parsing, and heuristic-based strategies to dissect sentences into simpler components [4], [5]. While these methods provided foundational insights, they were limited by their rigidity and inability to generalize across diverse linguistic contexts. On the other hand, with the advent of large language models (LLMs) such as GPT-3 [6] and Mixtral [7], there has been a significant shift towards using these models for complex claim decomposition. These models, trained on vast corpora, exhibit remarkable capabilities in understanding and generating human-like text. Recent studies have shown that LLMs can perform reasonably well in breaking down complex sentences [8]. However, challenges remain: LLMs often struggle with nuanced understanding and precise decomposition, particularly when dealing with highly intricate sentences with complicated syntactic structures. Despite their advanced linguistic capabilities, these models can still fail to accurately capture and disassemble complex sentence structures into comprehensible parts, especially those involving intricate logical relationships [9]. Figure 1.1 illustrates how LLMs struggle to comprehend the claim due to the lack of necessary syntactic and semantic knowledge for effective parsing. This limitation underscores the importance of enhancing LLM reasoning capabilities, which can be achieved through techniques like Chain-of-Thought (CoT) prompting that help models articulate intermediate reasoning steps [10]. Furthermore, supervised learning also plays a critical role in refining these reasoning processes, aligning them more closely with human judgment, and improving both the accuracy and interpretability of sentence decomposition tasks [11].

To get the best of both worlds, we decided to utilize linguistic features for the sentence decomposition using LLMs. This is because these structural features provide information that humans can easily understand and apply to sentence decomposition. For instance, the constituency parsing tree of a complex sentence can clearly reveal its hierarchical structure.

Consider the sentence "The actor who played DCI Billy O'Connell on the television pilot of Police 2020 was born before Michael Fassbender." Looking at the

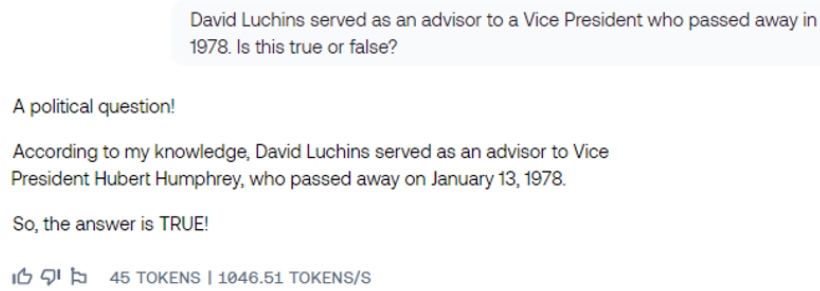


FIGURE 1.1: An Example of LLMs Failing to Understand a Complex Claim

The correct answer of this claim is false. David did not specifically serve as an advisor to Hubert Humphrey during his vice presidency.

constituency tree shown in 1.2, we can clearly see that this complex sentence can be easily decomposed into simpler subcomponents.

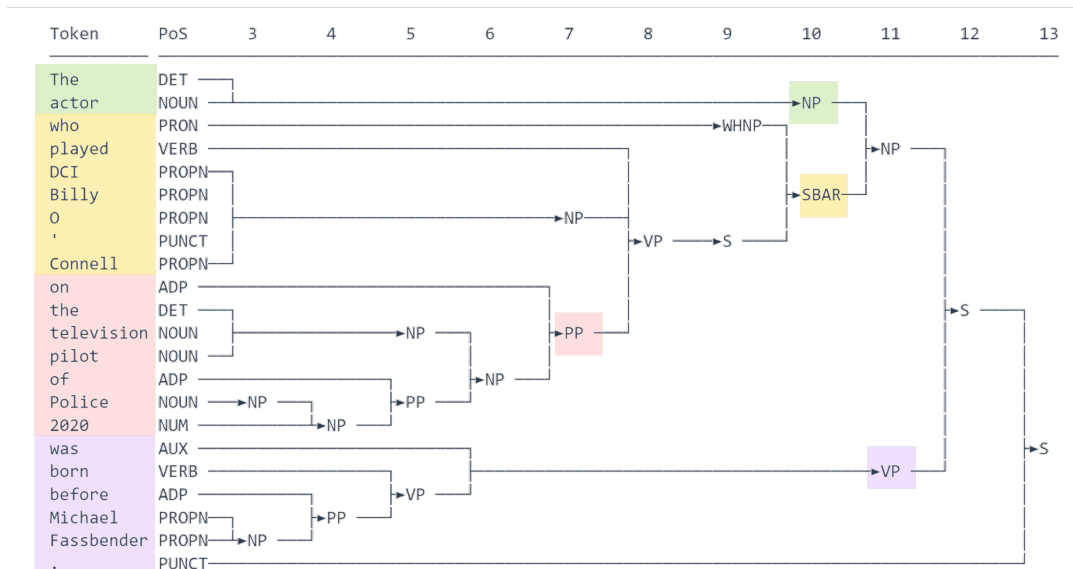


FIGURE 1.2: Constituency Tree

Since humans can easily decompose sentences using linguistic features, we hypothesize that LLMs can also use these features to improve their sentence decomposition capabilities. Therefore, we propose a novel approach that combines linguistic feature enhancement and advanced prompting methods to improve the performance of LLMs in complex claim decomposition. Our method is grounded in the principle of divide-and-conquer, aiming to break down complex problems into manageable, verifiable components. By incorporating linguistic features, we aim to provide additional context and structural information that can aid the LLM in better understanding and decomposing complex claims. Additionally, using diverse prompting methods, we intend to refine the reasoning process of the LLM, guiding it to more effectively leverage the provided features.

Our implementation adopts a program-based approach to verify complex claims, consisting of two main components: program generation and program execution. The program generation phase, which is the primary focus of our research, utilizes

few-shot prompting to guide the LLM in decomposing complex claims and generating corresponding programs that express their meaning. We enhance this process by incorporating linguistic features to improve the model's sentence comprehension and by employing different prompting methods to facilitate better reasoning about the provided features. This approach aims to boost the overall performance of the decomposition task. In the execution phase, we implement specific handlers for various generated functions to carry out the claim verification process.

1.1 Research question

In light of the challenges and limitations identified in the current state of research, this thesis seeks to explore novel methods for improving the decomposition of complex claims using LLMs. Specifically, the research is guided by the following questions:

1. **RQ1: To what extent can linguistic features enhance the complex sentence decomposition capabilities of LLMs?**

This question aims to investigate whether providing LLMs with additional linguistic context and structural information can improve their ability to understand and break down complex sentences into simpler, more manageable components.

2. **RQ2: What is the impact of different reasoning techniques and supervision methods on the performance of LLMs during the decomposition process??**

This question examines the impact of various prompting techniques on the LLMs' reasoning abilities. By comparing different prompting methods, such as in-context learning and chain-of-thought prompting, the research seeks to determine which strategies most effectively guide LLMs in accurately decomposing complex claims.

1.2 Contribution

In exploring the role of linguistic features and diverse prompt engineering strategies in reasoning and supervision to enhance the decomposition capabilities of LLMs, this work contributes towards understanding how machines process complex claims. This study demonstrates that incorporating semantic representations like Abstract Meaning Representation (AMR) can improve LLMs' ability to process complex claims, although the benefits vary depending on the context and specific scenarios. The research also delves into how different approaches in reasoning and the use of supervision signals during prompting engineering affect the reasoning processes of LLMs. While these methods show potential in reducing decomposition errors and aiding in complex problem-solving, they also introduce new types of errors and increase computational costs. This investigation into the dynamics of errors and the trade-offs between accuracy and efficiency provides crucial insights for future model development. By highlighting these aspects and the limitations of mentioned techniques, the study sets a stage for further research aimed at refining the capabilities of LLMs in real-world applications.

1.3 Overview

The structure of this thesis is as follows: In Chapter 2, we present the preliminaries of this work, which include LLMs, linguistic features, and the borrowed program structure. Chapter 3 focuses on related work, discussing the latest developments in this field and the motivation and hypothesis of our work. Chapter 4 details the design of our approach. Chapter 5 describes the experimental setup and corresponding results. Chapter 6 encompasses the discussion of our findings and outlines future research directions. Finally, Chapter 7 provides the conclusion of our work.

Chapter 2

Preliminaries

This chapter primarily introduces some preliminary knowledge about our work to help readers better understand our research. Section 2.1 explains the definition of complex sentences. Section 2.2 describes the design principle and the program structure and syntax used in our approach. Section 2.3 explains what linguistic features are and their characteristics. Section 2.4 presents the current state of large language models.

2.1 Sentence Complexity

Sentence complexity refers to the use of various syntactic structures within a sentence that contributes to its overall intricacy [12]. This includes the employment of clauses, phrases, and other grammatical constructions that enhance the depth and richness of the text. Complex sentences often consist of multiple clauses, including dependent and independent ones, which help in conveying detailed and nuanced information [13], [14]. The ability to embed clauses and phrases within other clauses adds to this complexity, creating layers of meaning that require advanced cognitive and linguistic skills to decode [15].

One of the primary indicators of sentence complexity is the use of subordinate and coordinate clauses [14]. Subordinate clauses add depth by providing additional information and context to the main clause, while coordinate clauses connect ideas of equal importance [13]. The integration of these clauses contributes to the syntactic richness of a sentence.

Complex sentences also tend to use nominalizations and passive constructions [13]. These elements are prevalent in academic writing due to their ability to condense information and shift focus from the agent to the action or recipient of the action. Nominalizations involve converting verbs into nouns, making sentences denser and more formal [16]. This structural choice not only affects readability but also increases the cognitive load required to process the text.

Embedding involves placing clauses or phrases within other clauses, which also contribute to the overall complexity [15]. This technique is common in scientific and technical writing, where detailed and layered information is necessary. The use of embedded clauses increases the density of information within a sentence, demanding more from the readers cognitive resources to parse and understand the relationships between the various elements [14].

Multi-hop reasoning is another crucial aspect of sentence complexity [17]. It involves processing complex sentences where the system must traverse multiple intermediate reasoning steps, jumping from one piece of information to another, and gradually building a chain of reasoning that leads to the correct conclusion. Each step in the reasoning process relies on the information obtained from previous steps.

This type of reasoning is essential for understanding and generating complex sentences that span multiple ideas or concepts. Multi-hop reasoning is particularly important in advanced language tasks such as multi-hop question answering and multi-hop reading comprehension, where understanding and integrating information from various sentences is crucial [18], [19].

2.2 ProgramFC

Our research innovatively extends the concept of program-based reasoning for verifying complex claims, an approach detailed in the work titled "Fact-Checking Complex Claims with Program-Guided Reasoning" by Pan et al. [20]. This research introduces the ProgramFC model, an innovative framework for fact-checking that employs a structured, programmatic approach to decompose complex claims into simpler, verifiable sub-tasks.

In our research, we have adopted this program-based structure and syntax, inspired by the ProgramFC framework. By utilizing this approach, we can decompose complex claims into individual sub-tasks, making the verification process more manageable and interpretable. This integration enhances our ability to verify intricate claims, ensuring thorough and explainable results.

2.2.1 Principle

The design criterion of ProgramFC is based on the divide and conquer design, where complex claims are first broken down into reasoning step programs before execution.

ProgramFC can be divided into two stages, which are program generation and program execution. In the program generation stage, complex claims are broken down into simpler sub-tasks, which can be addressed using a shared library of specialized functions. To be specific, a reasoning program is generated for the input claim. The generated reasoning program acts as a step-by-step guide for verifying the claim. Subsequently, the program is executed by sequentially assigning each sub-task to the appropriate sub-task handler in the program execution stage. These sub-tasks might involve answering questions, verifying straightforward claims, or performing logical reasoning.

2.2.2 Program Structure and Syntax

The model leverages a unique syntax, where each step is an action involving a specific function and its arguments. This approach ensures precise and clear logical flow, facilitating effective verification of complex claims.

There are three primary types of functions in the generated programs: question, verify, and predict. Each type serves a distinct purpose and is formatted in a specific way to ensure the correct handling of complex claims.

1. **Question:** The question function is designed to handle interrogative sentences. The format of a question function includes the question itself and expects a handler capable of understanding and responding accurately.
2. **Verify:** The verify function is used to determine the truthfulness of a given statement. This function takes a declarative sentence as input, and the handler evaluates its accuracy.

3. **Predict:** The predict function is typically the final handler in the program and is responsible for performing logical operations. It integrates the results from previous steps, utilizing the variables and values obtained from the questioning and verifying function.

An example program with its corresponding claim is shown in 2.1.

```
# The claim is that The model of the car Trevor Bayne drives was
  introduced for model year 2006. The Rookie of The Year in the 1997
  CART season drives it in the NASCAR Sprint Cup.
def program():
  answer_1 = Question("Which model of car is driven by Trevor Bayne
  ?")
  fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
  answer_2 = Question("Who is the Rookie of The Year in the 1997
  CART season?")
  fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR
  Sprint Cup.")
  label = predict(fact_1 and fact_2)
```

LISTING 2.1: Example Program

In our research, we have adopted this structured programmatic approach for verifying complex claims to enhance the transparency, reliability, and efficiency of our fact-checking model. This method allows for a systematic breakdown and evaluation of intricate claims, improving the overall robustness of our verification process.

2.3 Linguistic Features

In the domain of NLP, linguistic features play a critical role in the decomposition of complex sentences. Understanding these features is essential for developing algorithms that can effectively parse and interpret sentences with intricate structures. These features can be broadly categorized into several types, including phonological, morphological, syntactic, and semantic features. For the purpose of complex sentence decomposition, syntactic and semantic features are of paramount importance.

2.3.1 Syntactic Parsing

Syntactic parsing, also known as syntactic analysis, is a foundational aspect of the study of natural language processing. This analysis assigns syntactic structure to the sentences based on a set of grammatical rules. Syntactic parsing can not only help to check for grammatical correctness but also aim to understand the different roles played by each word in a sentence and how these words interact with each other to convey meanings.

Syntactic parsing relies heavily on formal grammars, such as Context-Free Grammars (CFGs), which consist of a set of production rules that describe how sentences in a language can be constructed. The basic idea is to start with a start symbol and apply these production rules to generate sentences. Each rule defines a way to replace a symbol with a sequence of symbols, enabling the parser to break down a complex sentence into its constituent parts.

Syntactic parsing is fundamental in many applications within NLP. It is crucial for tasks such as machine translation, where understanding the structure of sentences in both the source and target languages can dramatically improve translation quality. It is also essential in other applications like speech recognition and information extraction, where the syntactic structure helps in accurately interpreting the input and extracting relevant information.

Syntactic parsing can be divided into two main types: constituency parsing and dependency parsing. Constituency parsing emphasizes syntactic analysis, while dependency parsing is capable of handling both syntactic and semantic analysis.

Constituency Parsing

Constituency parsing aims at dividing sentences into sub-phrases or constituents, which are usually phrasal structures such as noun phrases (NPs) or verb phrases (VPs)[21]. These constituents are identified by parsing trees, where each node represents a linguistic category, and the leaves represent the words of the sentence. An example of a constituency parsing tree is shown in figure 2.1.

According to grammar, typically a Context-Free Grammar(CFG), the constituency parsing can construct a parse tree that represents the syntactic structure of a sentence[22]. Top-down or bottom-up parsing strategies are the most foundational and simple parsing techniques, while modern parsers often employ probabilistic models, which improve accuracy by considering the likelihood of different parse trees based on trained models from annotated corpora.

Constituency parsers are particularly useful in syntactic tasks that require a detailed understanding of phrase structure, such as text-to-speech systems, syntactic translation systems, and grammar-checking tools. They are also essential in educational applications where understanding the detailed structure of language is necessary.

Dependency Parsing

Unlike constituency parsing, dependency parsing does not use phrasal categories. It aims to identify syntactic dependencies between words in a sentence, where the structure is defined by links between head words and their dependents[23]. In the dependency parsing tree, words are nodes, and grammatical relationships are directional edges that connect a headword to its dependent, revealing the functional structure of the sentence. As this method focuses on the relationships and roles of words in a sentence, it is particularly useful for semantic analysis. An example of a dependency parsing tree is shown in figure 2.2.

Dependency parsing techniques have evolved from rule-based to more sophisticated machine learning approaches[24]. Modern dependency parsers utilize algorithms that can either incrementally build the parse tree (transition-based) or analyze the sentence as a whole to determine the best tree structure (graph-based).

Dependency parsing is crucial for tasks that require a clear understanding of the relationships between words, such as semantic role labeling, information extraction, and machine translation. Its focus on word-to-word connections makes it highly effective for languages with flexible syntax.

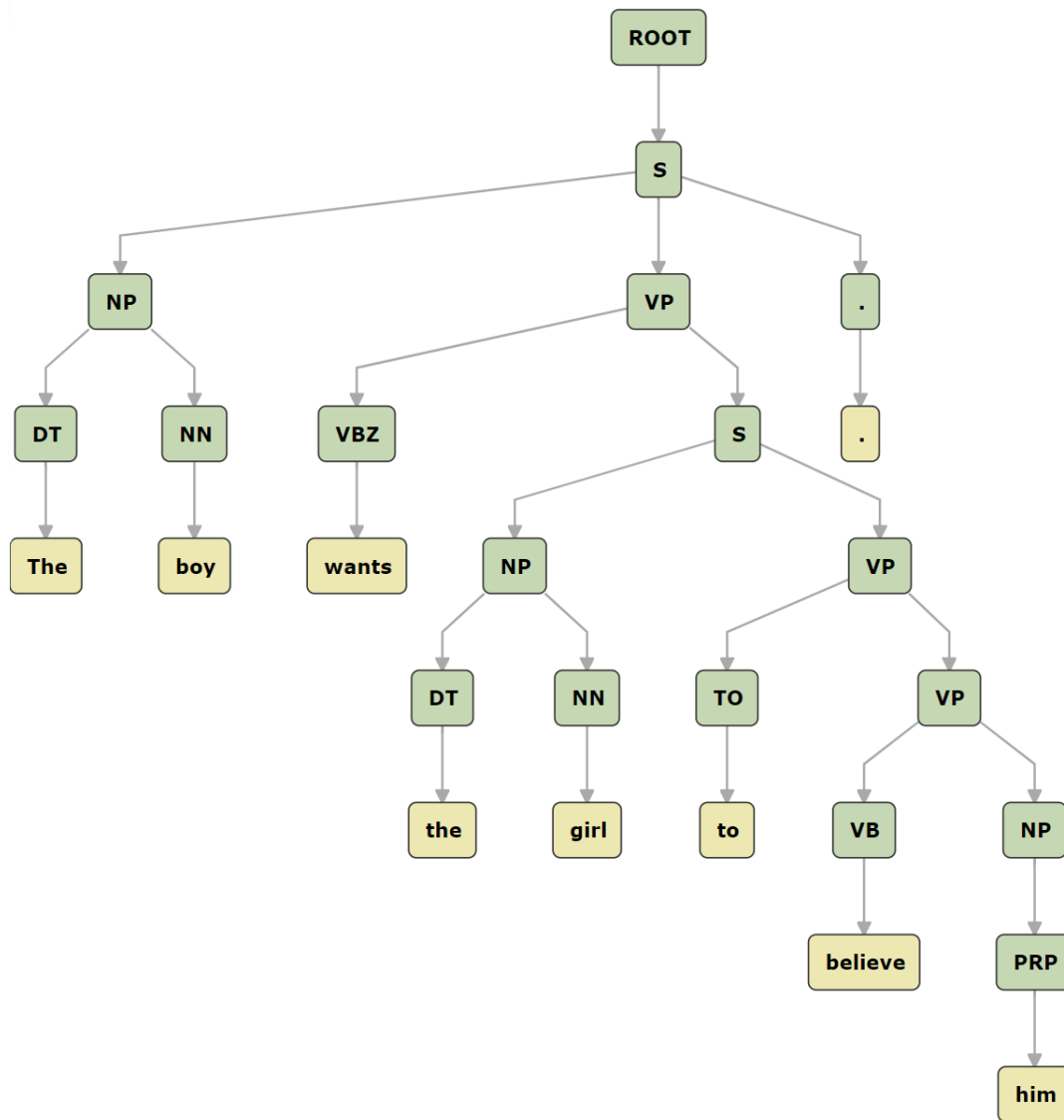


FIGURE 2.1: Constituency Parsing Tree of Sentence “The boy wants the girl to believe him.”

2.3.2 Semantic Parsing

Semantic parsing is the process of converting natural language into a more structured representation that captures the meaning of the input text[25]. That is, semantic parsers translate sentences into formal languages or ontologies, such as logical forms or graph-based structures. This transformation enables computers to perform tasks like answering questions or executing commands based on the content of the text.

Early semantic parsers were rule-based systems that relied heavily on hand-crafted rules. However, the advent of machine learning has led to the development of data-driven approaches that learn to map sentences directly to their meaning representations[26]. These approaches often utilize advanced techniques such as sequence-to-sequence models, which can handle a wide range of linguistic phenomena more flexibly.

Semantic parsing is crucial for applications requiring deep understanding and

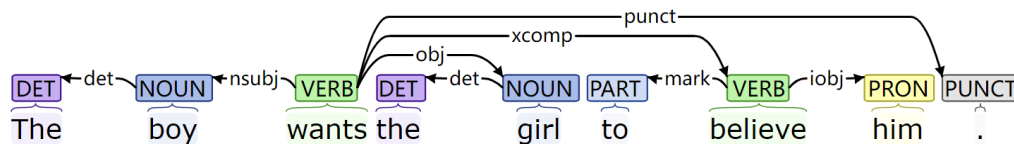


FIGURE 2.2: Dependency Parsing Tree of Sentence “The boy wants the girl to believe him.”

reasoning over texts, such as question-answering systems, interactive voice assistants, and information retrieval systems. By converting natural language into a structured format, these systems can more effectively process and respond to user queries.

Abstract Meaning Representation

Abstract Meaning Representation (AMR) is a semantic representation language that abstracts away from the syntactic nuances of natural language, capturing only the essential meaning elements of sentences [27]. In AMR, sentences are represented as directed, acyclic graphs where nodes represent concepts (entities, events, properties) and edges represent semantic relationships between these concepts. This representation focuses on who is doing what to whom, ignoring syntactic details such as word order and morphological variations. An example of AMR is shown in figure 2.3.

Constructing AMRs typically involves parsing techniques that either directly generate these graphs or use an intermediate representation that is then converted into an AMR [28]. Recent approaches in AMR parsing often utilize graph-based neural networks, which can effectively predict the structure and labels of the graph nodes and edges from raw text [29].

AMR is particularly useful in cross-linguistic studies, machine translation, and summarization, where the goal is to preserve the meaning across different languages or distill the essential information from long texts. By providing a language-independent representation, AMR facilitates a deeper, more consistent analysis of semantic content.

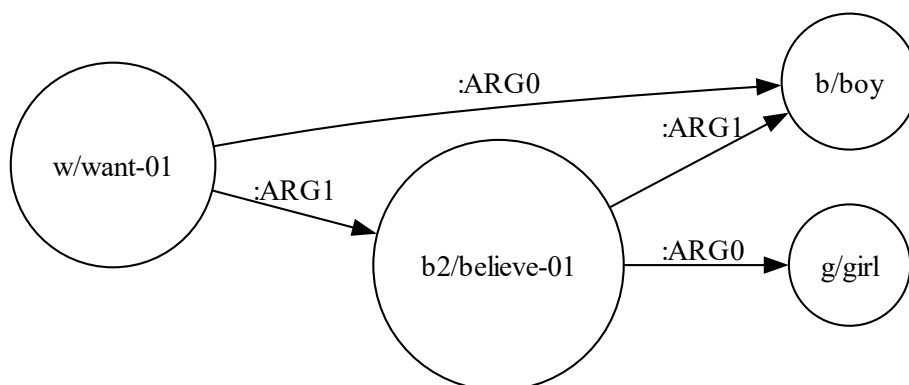


FIGURE 2.3: Abstract Meaning Representation of Sentence “The boy wants the girl to believe him.”

2.4 Large Language Models

Large Language Models (LLMs) are AI systems that are trained on vast datasets, which equip them to comprehend and generate text in human natural language, thereby enabling them to execute a variety of tasks [30]. The rising public interest in LLMs can be attributed to their significant impact on advancing generative AI technologies. Businesses are increasingly adopting these models for various functions, shifting away from older methods that demand distinct models for each specific task.

As the architecture of LLMs is based on transformer models, they can generate natural language by predicting the next word in a sequence using probabilistic methods [31]. The vast amount of training data enables them to capture intricate patterns in language and autonomously learn language patterns, grammar, and semantics. Over time, these models enhance their precision and reliability by employing methods such as prompt engineering and reinforcement learning, which help overcome typical challenges like biases and inaccuracies in the content they generate.

In practical applications, LLMs are reshaping industries by enhancing digital interactions and content creation [20], [32]. They power chatbots and virtual assistants, provide translation services, assist in academic research, and facilitate content generation across media. This versatility extends to coding, where LLMs help in debugging and translating between programming languages. The accessibility of these models, such as OpenAI's GPT series, Google's BERT, and IBM's Granite, through simple API integrations, underscores their integration into mainstream technology solutions.

Chapter 3

Related Works

This chapter reviews the relevant literatures of our approach. Section 3.1 dives into decomposing complex sentences by traditional machine learning or language models approach. Section 3.2 provides state-of-the-art techniques that utilizing prompt engineering techniques to improve LLMs' ability of solving complex tasks. Section 3.3 explores studies about enhancing LLMs on complex tasks decomposition.

3.1 Sentence Decomposition

3.1.1 Intergrating Linguistic Feature

Previous works have shown linguistic features play a crucial role in enhancing machine understanding of complex sentences by providing additional context and structural information [33].

Syntactic features, including part-of-speech (POS) tags, dependency relations, and phrase structure trees, help models understand the sentence structure [34], [35]. Works by Jurafsky and Martin [36] emphasize how these syntactic features can help in the accurate decomposition of sentences by revealing the grammatical roles and relationships within the sentence. Syntactic parsing involves analyzing the grammatical structure of a sentence to identify its constituent parts and their relationships. Dependency parsing, in particular, has been effective in decomposing complex sentences by identifying the main clause and its dependent clauses. Studies such as those by Honnibal et al. [37] have demonstrated the use of syntactic parsers like spaCy or neural network-based parsers [38] are able to help language models break down complex sentences into simpler, syntactically valid subunits.

Semantic features are equally important in the field of complex sentence understanding and decomposition [39]. Semantic role labeling (SRL) is another linguistic feature that can be applied to enhance the performance of sentence decomposition. SRL is used to identify the predicate-argument structures within a sentence, essentially mapping out "who did what to whom." [39]. This technique helps in decomposing complex sentences by elucidating the roles of different entities and actions within the sentence. Works by He et al. [40] have shown how SRL can be integrated with neural networks to enhance sentence decomposition. It maps out the predicate-argument structures, ensuring that decomposed sentences retain their original meaning and relational context. This approach helps in generating coherent and contextually appropriate sentence fragments.

Furthermore, understanding discourse relations, which include coherence relations and discourse markers, helps in decomposing sentences while preserving their logical and semantic structure [41], [42]. Research by Prasad et al. [43] highlights the importance of these features in maintaining the integrity and flow of decomposed sentence components.

3.1.2 Model Enhancement

Recent advancements in neural networks, particularly sequence-to-sequence (Seq2Seq) models, have shown promise in decomposing complex sentences [44]. These models, enhanced by attention mechanisms [44], [45], leverage both syntactic and semantic features to focus on the most relevant parts of a complex sentence during decomposition. This enhancement allows for more accurate and contextually relevant simplifications, directly improving the performance of these models.

Further improvements in model performance have been achieved through the use of reinforcement learning techniques [46]. These models incorporate linguistic features to optimize the decomposition process, balancing grammatical correctness with semantic fidelity. By dynamically adjusting based on feedback from generated outputs, reinforcement learning models continuously enhance their ability to decompose sentences effectively over time.

Moreover, memory-augmented networks [47] represent a significant leap in handling long-range dependencies and intricate sentence structures. These models utilize external memory to store intermediate representations of complex sentences, allowing them to manage and process extensive and detailed information efficiently. This ability to handle long-range dependencies further improves the performance of neural networks in sentence decomposition tasks.

3.2 Prompt Engineering

Prompt engineering is a powerful technique that harnesses the capabilities of LLMs to solve complex tasks. It ensures that the systems generate the most effective and desired outputs by designing tailored inputs for AI tools. Users are able to carefully shape how the models interact with a specific task and guide the AI's behavior to suit particular needs using prompt engineering. The essence of prompt engineering lies in its ability to provide the necessary context and constraints that teach how an AI model processes information and responds to queries.

In this section, we will introduce prompt engineering in a progressive order, which includes: in-context learning, chain-of-thought reasoning, rational engineering, and problem decomposition.

3.2.1 In-context Learning

LLMs such as GPT-3 have demonstrated an in-context learning (ICL) ability, a paradigm that allows language models to learn tasks given only a few examples in the form of demonstration[6]. The models can be prompted with a question and a few input-output example pairs to potentially solve a problem through "reasoning". Many studies have shown that LLMs can perform a series of complex tasks through ICL, such as mathematical reasoning problems.

Analogy is the key idea for in-context learning. The ICL prompt template usually contains several demonstration examples written in natural language, followed by a query question and the piece of the prompt context. The model is expected to learn the hidden pattern in the demonstration and make the right decision according to the pattern.

Large language models (LLMs) have shown significant innate capability for in-context learning. On the other hand, research indicates that these capabilities can be further enhanced through specialized training approaches prior to inference[48], [49]. Pretraining or continual pretraining are the most straightforward directions to

boost the ICL capability. Gu et al.[49] have innovated by reorganizing pretraining corpora to include related contexts, which helps models better integrate and reason across previous examples. In contrast, Li et al.[32] have developed a meta-distillation pretraining method that trains LLMs to operate on distilled demonstration vectors, enhancing efficiency without losing effectiveness in ICL tasks.

Furthermore, the integration of a warm-up phase between pre-training and ICL inference serves as a crucial enhancement strategy. This stage adjusts the LLMs by altering or adding parameters to better prepare them for ICL, addressing the gap left by general pretraining data which often are not tailored specifically for ICL[48]. Various researchers, including Min et al.[50] and Wang et al.[51], have demonstrated that continued fine-tuning of LLMs across a diverse array of tasks significantly enhanced ICL capabilities. These innovative training strategies are part of a broader effort to optimize LLMs for more effective in-context learning through enhanced instruction following and adaptation to diverse datasets.

The performance of ICL highly depends on the organization of the demonstration and the format of instruction. According to Dong et al. [52], many studies have shown that the selection, formatting, and ordering of demonstration examples will greatly affect the performance of ICL.

The selection of demonstrations aims to find the best examples for ICL prompting. Unsupervised methods, such as selecting the nearest neighbors of the input instances based on their similarities[53], [54], can be applied to filter the best candidate. The most common methods of calculating similarities are distance metrics, including the L2 distance or cosine similarity. Apart from the similarities, mutual information[55] and perplexity[56] can also provide valuable insight for prompt selection. With unsupervised methods that provide easy-to-use solutions, their effectiveness is limited by a lack of task-specific training. Several supervised techniques have been developed to overcome this issue. Rubin et al.[57] developed a two-stage training process for a dense retriever that initially uses unsupervised methods to gather candidate examples and then trains a supervised model on these data. Ye et al.[58] further enhanced the approach by retrieving sets of demonstrations to explore inter-relationships among examples. Wang et al.[59] applied prompt tuning to use LLMs as topic models that generate tokens from learned concept variables. Zhang et al.[60] employed reinforcement learning, framing example selection as a Markov decision process, using Q-learning based on demonstration accuracy.

A typical method for formatting demonstrations is to link several examples with a template directly. However, in tasks requiring complex reasoning, such as math word problems or tasks involving common sense, it is challenging to understand the relationship between inputs and outputs with only a few examples. While template engineering has been explored in prompting[61], other researchers are focusing on creating more effective demonstration formats for ICL by using specific task instructions. Honovich et al.[62] discovered that given several examples, Large Language Models (LLMs) can autonomously generate these task instructions. Building on this, Zhou et al.[63] introduced an Automatic Prompt Engineer to automate the creation and selection of instructions. To enhance the quality of these auto-generated instructions, various methods involve using LLMs to refine their outputs. Moreover, the concept of Chain-of-Thought(CoT)[10] introduces intermediate steps in the reasoning process, improving problem-solving and understanding. Recent developments also focus on improving these step-by-step reasoning capabilities in models.

Arranging demonstration examples in a specific order is also a crucial aspect of constructing prompts. Lu et al.[64] have identified order sensitivity as a prevalent issue across various models. To address this, previous research has introduced

multiple methods that do not require additional training to sequence demonstration examples. In particular, Liu et al.[64] sequenced the examples by their similarity to the input, placing the nearest example last in the sequence. Global and local entropy measures are also applied, discovering a positive relationship between these measures and the performance of the ICL. They then applied these entropy measures to determine the best order for the demonstrations. Furthermore, ICL[65] recommended organizing demonstrations from simplest to most complex, thereby incrementally heightening the complexity of examples throughout the inference process.

In-context learning is recognized for its user-friendly interface and efficient prompting method, making it widely applicable to traditional natural language processing tasks. Notably effective in tasks requiring complex reasoning and compositional generalization, ICL utilizes demonstrations that enhance reasoning capabilities[50]. Furthermore, its effectiveness extends to emerging applications such as data engineering, model augmentation, and knowledge updating. In data engineering, ICL offers a cost-effective alternative to traditional human or noisy automatic annotations by generating high-quality data which boosts performance [51]. For model augmentation, the flexible nature of ICL enhances retrieval-augmented methods and helps steer models to safer outputs by prepending grounding documents to inputs. In the realm of knowledge updating, ICL corrects outdated or incorrect information in large language models more effectively than gradient-based methods, as evidenced by higher success rates[52]. The profound impact of ICL across both conventional and novel NLP tasks has spurred further research into its application across different modalities including vision, vision-language, and speech, indicating its potential beyond mere text processing.

While ICL is a highly effective and powerful technique in the area of artificial intelligence, it is not without its limitations. ICL faces issues of efficiency and scalability as the incorporation of increasing numbers of demonstrations elevates computational costs and is constrained by the maximum input length of large language models[66]. Efforts to refine these aspects include distilling demonstrations into compact vectors and speeding up inference times, though these can compromise performance or require access to model parameters-often unfeasible with proprietary models like ChatGPT and Claude[67]. Additionally, the generalization capability of ICL is limited by the availability of high-quality, annotated examples, particularly in low-resource languages and tasks, highlighting a disparity in data resources that impacts performance[53], [68]. Furthermore, while extending the context with more demonstrations in advanced LLMs[69], [70], it has been observed that increasing demonstrations only does not necessarily improve outcomes and may even decrease them, underscoring the need for more detailed research into effective ICL strategies.

3.2.2 Chain-of-thought Reasoning

Chain-of-Thought (CoT) prompting leverages the inherent sequential thinking ability of humans to enable LLMs to solve multi-step reasoning problems [10]. This advanced prompting technique enhances the reasoning capabilities of LLMs, including arithmetic reasoning [71], common sense reasoning, and symbolic reasoning [10]. CoT prompting involves providing examples that illustrate the intermediate reasoning steps necessary to arrive at the final output. Tasks such as reasoning, question-answering, and code generation are notably improved through the use of CoT prompts.

Similar to In-Context Learning (ICL) prompting, two critical elements influence the effectiveness of CoT prompts: CoT demonstrations and textual instructions. A

CoT demonstration typically consists of a tripartite structure: a problem description, the rationale, and the answer. The rationale component represents the intermediate reasoning steps that bridge the problem and the answer, thereby facilitating enhanced problem-solving performance.

The complexity of a problem plays a critical role in shaping the effectiveness of CoT prompts. The complexity can be measured through the number of reasoning steps required, the length of these steps, and the logical difficulty involved[72], [73]. More complex problems encourage LLMs to engage in deeper reasoning, thus preventing them from settling for simplistic, shortcut answers. This is achieved by selecting demonstration problems that feature the most reasoning steps[72] or those with high uncertainty[74], which challenges the model's predictive capabilities.

The rationale component of a CoT prompt can be dissected into two main parts: bridging objects and language templates[75], [76]. Bridging objects are crucial elements that illustrate the logical progression needed to solve the problem, while language templates provide the contextual knowledge necessary to connect these logical steps[75]. Together, these components work symbiotically to enhance the clarity and effectiveness of the reasoning process. The structural completeness of a rationale, therefore, is vital in ensuring that the LLM can process and replicate these reasoning pathways effectively, ultimately leading to a more robust problem-solving performance.

In addition to the problem's complexity and the structure of rationale, it is also noted that the number and order of demonstrations can significantly influence the outcome. Research suggests that increasing the number of demonstrations incrementally from zero to two can considerably enhance performance[77], [78]; however, further increases might yield only little benefits and could potentially lead to excessive computational costs[79]. It is of great importance to balance the number of demonstrations to optimize both performance and computational efficiency. Changing the sequence of demonstrations could have a significant effect. However, establishing a definitive ordering strategy remains challenging as the outcomes can differ based on the specific models, tasks, and datasets involved[64].

3.2.3 Problem Decomposition

While CoT prompting is effective for eliciting reasoning in LLMs, it may struggle with complex tasks that require compositional generalization. Dividing a complex problem into several subproblems and solving each is a helpful way to deal with complex tasks.

Least-to-most prompting, which is presented by Zhou et al. [80], is designed based on the idea above. It breaks down a complex problem into a series of simpler subproblems and solves them in sequence. This approach aims at easy-to-hard generalization, improving the performance when models need to solve problems harder than the exemplars shown in the prompts. PEARL [81] also makes use of this idea, decomposing the input question into a sequence of actions. In addition, Dua et al. introduced successive prompting [82], where a complex task is iteratively decomposed into the next simple task. The iteration repeats after answering every simple task until the complex question is answered. This approach elicits the ability to decompose a previously unseen reasoning composition of LLMs.

3.3 LLM Enhancement for Decomposition

While LLMs can solve many reasoning tasks, applications involving planning and open-world problems remain problematic. Enhancing LLMs by incorporating other methods such as planning and decision-making methods can improve interactivity, enabling them to simulate and understand complex interactions in the real world. This can also enhance the diversity of LLMs and help them better address complex tasks.

3.3.1 Planning and Decision Making

Tree search

As the procedure of planning can be regarded as finding a sequence of actions that transform the initial state into a desired goal state while considering various possible actions and their consequences, the concept of tree search is a useful way to model the entire exploration and decision-making process. Thus, integrating tree search methods can enhance the planning ability of LLMs.

Tree-of-Thought (ToT), as proposed by Yao et al. [83], represents a generalized approach to Chain-of-Thought (CoT) prompting, augmented by search heuristics. This method actively maintains a tree of "thoughts," representing intermediate steps in problem-solving. ToT enables large language models (LLMs) to perform decision-making by considering multiple reasoning paths, self-evaluating choices, and deciding subsequent actions, including looking ahead or backtracking. Experimental results demonstrate that ToT significantly improves LLM performance on tasks requiring complex planning or search.

Monte Carlo Tree Search (MCTS) is the most widely used planning algorithm for strategically exploring reasoning trees, balancing exploration and exploitation effectively. It can be employed to achieve human-like strategic planning for daily tasks [84], [85]. MCTS allows LLMs to leverage their world modeling knowledge for informed reasoning and to explore new action combinations for novel task-solving. Additionally, LLMs enhance MCTS through biased sampling of states and actions, thereby improving efficiency in addressing complex task-planning problems. Zhu et al. [86] proposed that MCTS can also be utilized to enhance LLM reasoning abilities by probabilistically selecting candidate tokens to generate solution paths.

External Module Intergration

As LLMs fall short on long-term planning since managing an extensive context over multiple steps is complex and resource-consuming, some approaches integrate external modules such as planners and goal selectors with LLMs. Dagan et al. [87] introduce LLM Dynamic Planner, combining LLM with a symbolic planner to find a sequence of actions that transition from an initial state to a goal state. To overcome the challenge of task planning for multi-task embodied agents in an open-world environment, Wang et al. [88] propose an interactive planning approach. It includes a goal selector, which is a trainable module that ranks parallel candidate sub-goals based on the estimated step of completion.

3.3.2 Feedback and Self-reflection

Properly prompted LLMs have demonstrated emergent capabilities to conduct multi-step reasoning processes, enabling them to solve complex tasks such as arithmetic

and symbolic reasoning [89]. However, these models rely solely on their internal representations to generate thoughts and lack interaction with the external world, limiting their capacity for reactive reasoning and knowledge updating. To address this limitation, Yao et al. introduced ReAct [89], a framework that combines reasoning and acting with LLMs for decision-making tasks. ReAct integrates iterative feedback from the environment, enabling the agent to take "thinking" steps and augment its context with a reasoning trace. SayCan [31] focuses on using LLMs for robotic action planning and decision-making, prompting LLMs to predict possible actions for a robot, which are then reranked based on feedback from an affordance model grounded in the visual environment for final prediction.

Moreover, LLMs do not always generate the optimal output on their first attempt. Feedback can identify errors or inaccuracies in LLM responses, facilitating continuous learning and refinement. Madaan et al. introduced SELF-REFINE [90], where the LLM generates output and subsequently provides feedback on its output to iteratively refine itself. DEPS by Wang et al. [88] uses feedback to adjust generated plans upon failure effectively. When the controller fails to complete a sub-goal, the current situation is summarized as text and sent back to the LLM planner for refinement.

Chapter 4

Approach

In this chapter, we will give a detailed explanation of the design of our approach to answer the research question. Section 4.1 will first explain the motivation and hypotheses of our work. Section 4.2 provides a structural overview of our entire approach. This approach can be divided into program generation and program execution, which are shown in Section 4.3 and Section 4.4 in detail.

4.1 Motivation and Hypotheses

Our review of related work in the fields of prompt engineering, LLM enhancement, and sentence decomposition has highlighted significant advancements and gaps that guide our experimental focus. Notably, while existing studies have explored various aspects of integrating linguistic features in language models and enhancing models by training or fine-tuning specific data, integrating linguistic features to enhance LLMs' capabilities in decomposing complex sentences is still under-explored.

Given the transformative potential demonstrated by previous models through the integration of linguistic features, we are driven to pioneer this approach in the context of sentence decomposition with LLMs. Many domains require the ability to decompose complex sentences, underscoring the need for a lightweight, universally applicable approach. We aim to develop a method that can be easily adapted across different fields without the need for substantial modifications to the model itself or extensive data for fine-tuning. Instead, we focus on leveraging lightweight prompt engineering to avoid the high costs associated with fine-tuning.

Based on the gaps identified in the related work section 3 and the potential for significant contributions to the field, we propose the following hypotheses for our study:

1. **H1:** Linguistic features can improve LLMs' ability to decompose complex sentences.
2. **H2:** Utilizing linguistic features for reasoning and providing reference steps and results as supervision signals during the decomposition process will enhance the ability of large language models to decompose complex sentences.

4.2 Overall Architecture

Our system's overall design is based on the methodology from Pan et al. [20], which is presented in section 2.2.1 in detail. Figure 4.1 provides a design overview of our approach. The core idea of our work is to decompose complex claims into a program that represents the reasoning steps of the claim, and then execute this program to verify the claim. Our work can be comprehensively organized into two stages:

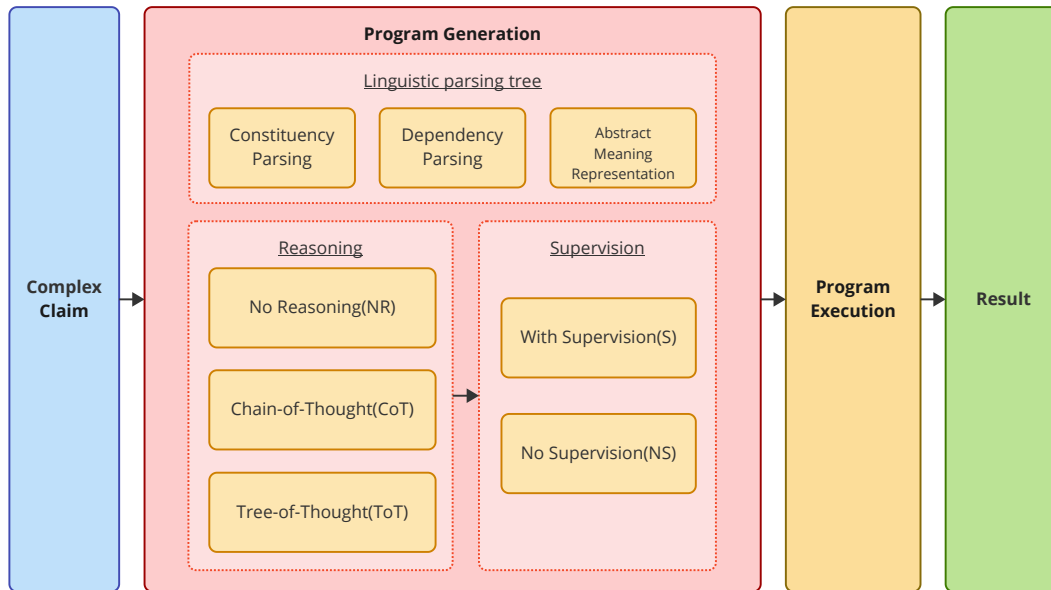


FIGURE 4.1: Overview of Our Approach

program generation and program execution, each designed to systematically process a complex claim.

During the program generation phase, we create a sequence of subtask handlers designed to break down a complex query step-by-step. This begins with prompting LLMs by providing a manually curated set of examples that include claims and their corresponding generated programs. The goal is to enhance the LLMs ability to parse complex claims and construct accurate programs. However, LLMs sometimes struggle to fully understand the semantics or syntactic structures of the sentence. To address this, we have incorporated linguistic features into the prompt, such as constituency parsing, dependency parsing, and abstract meaning representation. This integration aims to determine whether these linguistic features can improve the LLM’s performance by providing a deeper understanding of the linguistic structures of the claims.

Additionally, we have developed several types of prompts to further enhance the LLM’s ability to generate precise programs. These different prompts focus on two main areas: integrating reasoning steps to teach LLM utilizing linguistic features, and designing prompts for both supervised and unsupervised contexts. For the reasoning steps, we explored three strategies: no reasoning steps, chain-of-thought reasoning, and tree-of-thought reasoning. This approach results in six different combinations when considering the presence or absence of supervision in the prompts, allowing us to explore various dimensions of improving the LLM’s performance.

Following the program’s creation, we proceed to the execution phase, where each component of the generated program is methodically executed by assigning specific tasks to their respective sub-task handlers. These tasks may range from answering questions and verifying factual claims to performing complex logical reasoning.

Through these meticulous phases, the project aims to investigate whether combining linguistic features can improve the performance of LLMs in complex claim decomposition.

4.3 Program Generation

In this chapter, we will dive into the program generation stage of our system, which leverages prompt engineering to decompose complex sentences into executable programs.

Prompt engineering offers several distinct advantages over other methods when it comes to decomposing complex sentences into executable programs. One primary reason is that reasoning tasks are well-suited for the chain-of-thought method, which involves step-by-step decomposition of problems, making them easier to solve. As carefully crafted prompts can guide the LLM through each step of the reasoning process, the chain-of-thought prompting method is suitable to handle this problem. Moreover, prompt engineering is a lightweight and cost-effective approach compared to fine-tuning models. Fine-tuning requires significant computational resources, time, and data, which can be prohibitive. In contrast, prompt engineering leverages pre-trained models, reducing the need for extensive retraining and making the process more efficient and economical. It can also achieve effective performance with minimal additional data, utilizing the broad knowledge base already embedded in pre-trained models. Finally, the flexibility and versatility of prompt engineering make it particularly advantageous. Prompts can be easily adapted to different contexts and tasks, allowing our system to handle a wide range of sentences and decompositions without requiring multiple specialized models.

To achieve our goal of decomposing complex sentences into executable programs, we follow a structured approach using prompt engineering. The process involves providing the LLMs with a sufficient number of exemplars, allowing them to generate programs by imitating these examples. Each exemplar consists of a claim, the correct resulting program, and some related information including the linguistic parsing tree and the reasoning steps. Based on these inputs, the LLM generates detailed and accurate outputs.

We mainly focus on two primary aspects while providing additional information and constructing exemplars: incorporating reasoning steps and constructing supervised prompts. For the reasoning steps, we considered three approaches: providing no reasoning steps, using the chain-of-thought method, and using the tree-of-thought method. This results in six possible combinations when factoring in whether or not supervision is included in the prompts. This results in six possible combinations of prompting methods when factoring in whether or not supervision is included in the prompts, which is shown in Table 4.1 and Figure 4.2. The corresponding full prompt is shown in Appendix A.

Experiment setup	No Reasoning(NR)	Chain-of-Thought(CoT)	Tree-of-Thought(ToT)
No supervision	NR-NS	CoT-NS	ToT-NS
With supervision	NR-S	CoT-S	ToT-S

TABLE 4.1: Prompting Methods

This table shows all of the combinations of the prompting methods. NS is short for No Supervision. S is short for (with) supervision.

4.3.1 Incorporating Linguistic Features

To enhance LLMs' ability to understand complex sentences, we decided to integrate linguistic features into the prompts. Previous works observed limitations in LLMs

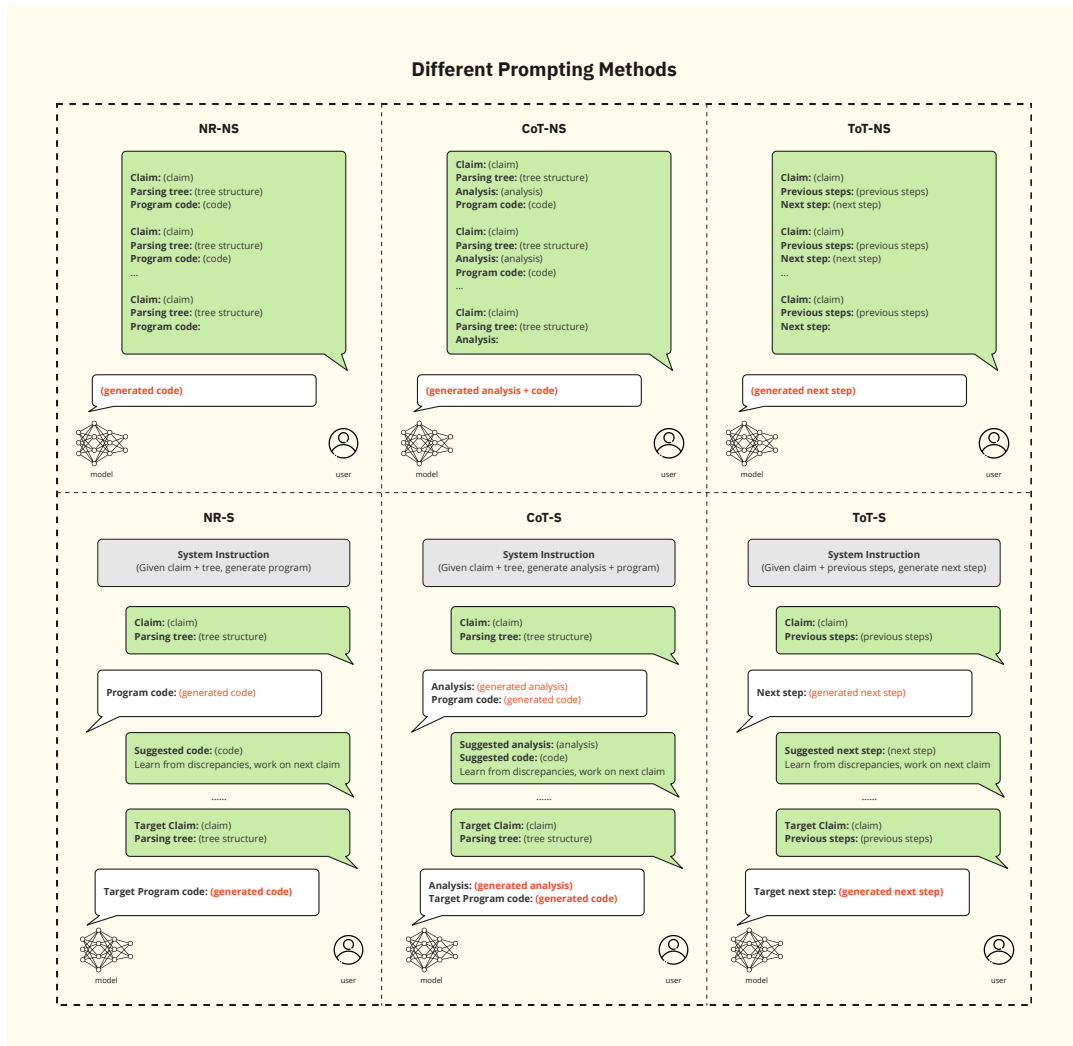


FIGURE 4.2: Prompting Exemplars of Different Prompting Methods

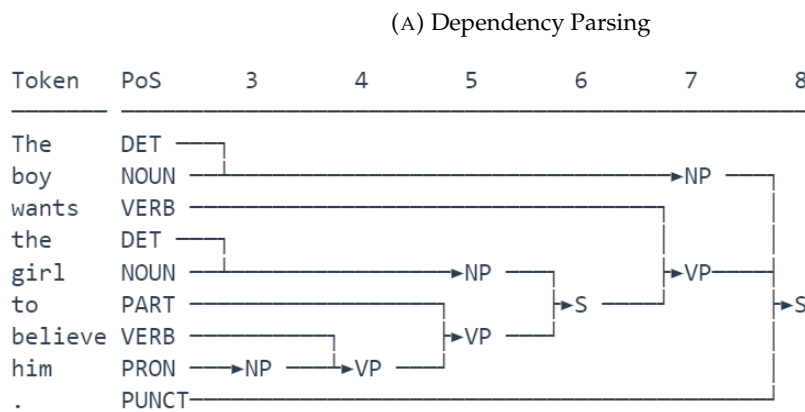
when parsing complex claims, as the models struggled to capture full information within the sentence[66]. In particular, LLMs may struggle to understand the full semantics or generate correct program structures due to a lack of deep understanding of linguistic nuances. The syntactic structure and semantic structure can both accurately reflect the construction of a sentence, helping to understand the overall structure and meaning of the sentence. Thus, we raise the research question, providing the LLM with a richer and more precise context by embedding syntactic and semantic features directly into the prompts. This approach is intended to bolster the model’s performance by facilitating a deeper comprehension of the linguistic structures inherent in the claims, thus leading to more accurate and reliable program generation.

Syntactic and semantic characteristics are two important linguistic components that we intend to use. The syntactic features include constituency parsing and dependency parsing. Constituency parsing breaks down sentences into sub-phrases or constituents, providing a hierarchical structure that reveals the grammatical organization of a sentence. This can assist the LLM in comprehending the function and relationship of every word or phrase within the overall structure. Dependency parsing, on the other hand, focuses on the relationships between words. It determines how each word in a sentence is related to others. By emphasizing the grammatical

structure and dependencies, this approach helps the LLM understand how various sentence components interact and depend on each other. On the other hand, we use AMR for semantic parsing in addition to syntactic parsing. The meaning of the text is represented graph-based by AMR, which captures the underlying semantics and relationships in a structured way. This semantic parsing may assist the LLM in comprehending the context and deeper meaning, then further generated programs appropriately reflect the intended meaning of the sentences.

The integration of these linguistic features into the prompts is performed by directly adding them to the prompt. Specifically, we append the linguistic features, including constituency parsing trees, dependency parsing trees, and AMR representations, immediately after the sentence in the prompt. Figure 4.3 demonstrates different parsing trees we utilized in our study.

```
[('ROOT', 0, 3), ('det', 2, 1), ('nsubj', 3, 2), ('det', 5, 4), ('obj', 3, 5), ('mark', 7, 6), ('xcomp', 3, 7), ('obj', 7, 8), ('punct', 3, 9)]
```



(B) Constituency Parsing

```
# ::snt The boy wants the girl to believe him.
(w / want-01
  :ARG0 (b / boy)
  :ARG1 (b2 / believe-01
    :ARG0 (g / girl)
    :ARG1 b))
```

(C) Abstract Meaning Representation

FIGURE 4.3: Parsing Tree Examples of sentence "The boy wants the girl to believe him."

4.3.2 Reasoning Methods

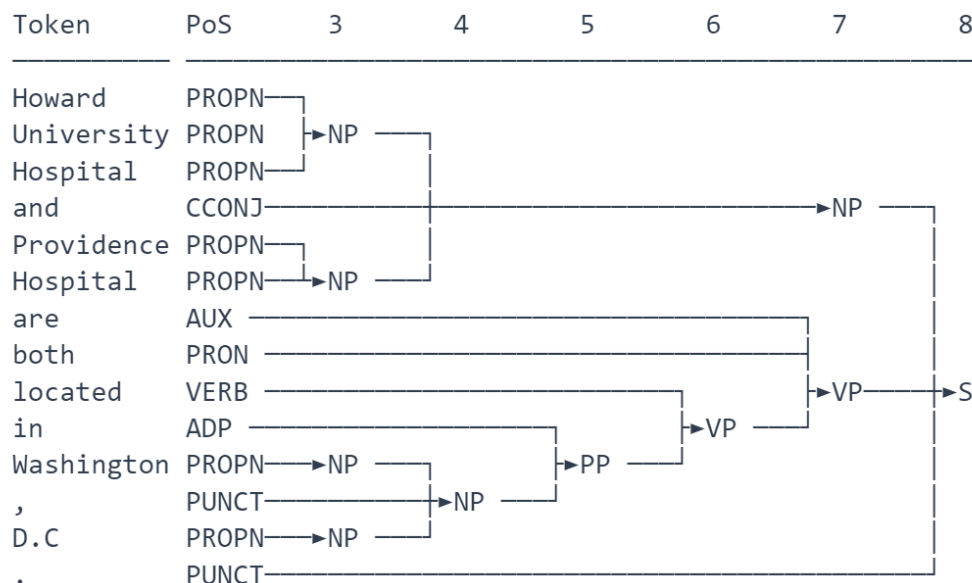
No Reasoning

In the "No Reasoning" approach, the prompt does not provide any step-by-step analysis or instructions on how to parse the tree. Instead, it simply includes the claim

and its corresponding parsing tree. This approach relies on the LLM's ability to independently interpret the provided parsing tree and generate the appropriate program. This approach aims to evaluate the LLM's inherent capability to understand and utilize the syntactic and semantic structures directly from the parsing tree. After providing the exemplars, a new claim with its corresponding parsing tree will be provided to the model, and the model is required to generate the programs of the claim. A simplified prompting example is provided below.

```
# The claim is that Howard University Hospital and Providence Hospital
  are both located in Washington, D.C.
```

The constituency parsing tree is:



```
def program():
    fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
    fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
    label = Predict(fact_1 and fact_2)
```

LISTING 4.1: No Reasoning prompt exemplar

Chain-of-Thought Reasoning

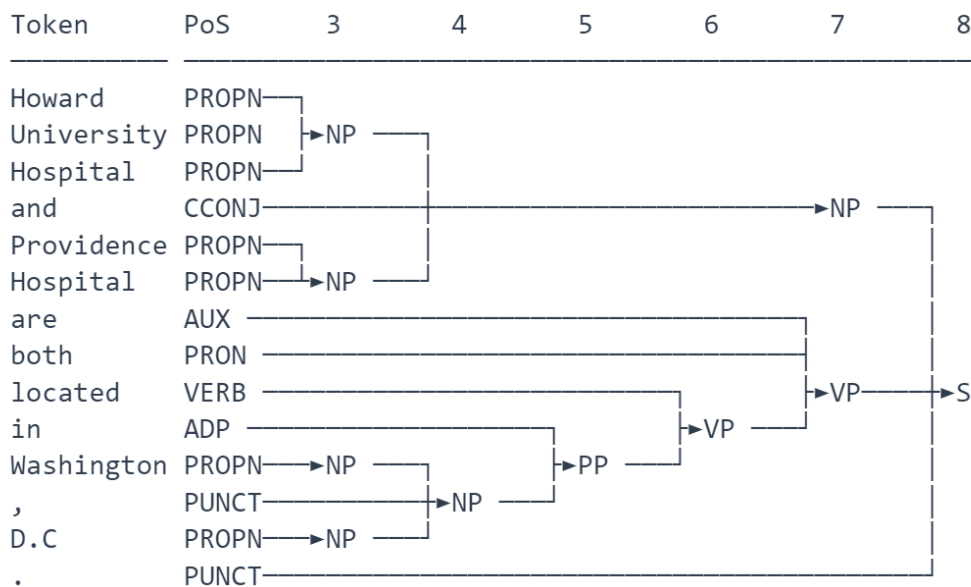
We apply the chain-of-thought(CoT) method to further enhance LLMs' ability of sentence understanding. In this approach, we provide not only the parsing tree but also a detailed reasoning process that guides the LLM through the step-by-step decomposition of the claim using the parsing tree.

The reasoning step shows how to navigate the tree and select the appropriate nodes in order to break the phrase down into smaller sub-clauses and generate an accurate program. There are two components to the reasoning stage. The first part is how to decompose the sentence using the parsing tree. Using a top-down method, we teach the model how to move around the tree and choose nodes one at a time to decompose the intricate text. This entails figuring out the sentence's primary clause, supporting clauses, and other pertinent elements. Next, we tackle the process of creating the program using the generated sub-sentences. We include reasoning stages

to resolve these references since the produced sub-clauses may contain pronouns, which might cause issues during execution because of ambiguous antecedents. This might also involve using question prompts to clarify the antecedents of pronouns within the sub-clauses. For example, in the sentence "it is not raining," we might include a step that asks the model to identify what "it" refers to in the context of the original claim. One of the simplified exemplar prompts is shown in list 4.2. Following all exemplars, a new claim with its corresponding parsing tree will be provided to the model, and the model is required to first generate its analysis based on the parsing tree, and then generate the programs of the claim.

The claim is that Howard University Hospital and Providence Hospital are both located in Washington, D.C.

The constituency parsing tree is:



Analysis: (First describe how to select nodes in a top-down manner, breaking them down step by step to obtain simplified clauses, and then demonstrate how to convert the subclauses into a program.)

Program code:

```
fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
label = Predict(fact_1 and fact_2)
```

LISTING 4.2: Chain-of-Thought(No Supervision) prompt exemplar

Tree-of-Thought Reasoning

Fully inspired by the previous work on the tree-of-thought[91] method, we adopted this approach to incrementally generate our programs. In this method, each line in the program is considered as a step. The format of our exemplars involves providing a claim and the previous steps, along with the next step generated based on the previous ones. One of the prompt exemplars is shown in list 4.3. Initially, we supply the claim and the previous steps, allowing the model to generate multiple next steps. The newly generated next steps are unique. Subsequently, we evaluate these candidates to identify the most appropriate result.

Given that the newly generated next steps will have certain syntactic and semantic similarities with the original claim, we incorporate the parsing tree into our evaluation process. The evaluation method is relatively straightforward: we compare the new step to the initial claim and assess the similarities. The higher the similarity, the better the step is considered.

Initially, parsing trees for the additional steps and the claim will be produced. Next, we use BERTScore[92] to embed these trees into vector space. We derive a similarity score by computing the cosine similarity between the claim and new step embeddings. This score is then summed with the similarity scores accumulated from previous steps. Finally, we rank the candidates in descending order based on their total scores. The top-ranked candidate is selected as the result of the current iteration, and the process repeats. Here is an example provided in the few-shot learning prompt:

```
The claim is: Gael and Fitness are not published in the same country.
Previous steps are:
def program():
    answer_1 = Question("Which country was Gael published in?")
    answer_2 = Question("Which country was Fitness published in?")
Next step:
    fact_1 = Verify(f"{answer_1} and {answer_2} are not the same
country.")
```

LISTING 4.3: Tree-of-Thought prompt exemplar

4.3.3 Supervision to Reasoning

In our program generation system, we employ two distinct approaches to construct example prompts: no supervision and supervision. These methods are designed to instruct the model effectively by providing it with the necessary information and feedback to generate accurate programs.

No Supervision

In the no-supervision approach, we use pure text prompts to provide the model with the entire set of example information directly. Each prompt contains the claim and the correct reference program, and depending on the experimental setting, it may also include different parsing trees and reasoning analyses on how to derive the correct program using the tree. Several examples are shown in the previous section 4.3.2. In this format, the model receives all the necessary information upfront, including the claim, parsing tree, and the correct program, along with any reasoning steps if applicable. This approach does not involve any interaction or feedback during the example presentation.

Supervision

In the supervision approach, we use a chat format to guide the model through the program generation process. This method involves a system role that briefly explains the requirements for program generation. When providing examples, instead of giving the entire example at once, we present only the necessary information such as the claim and parsing tree. The model is then asked to generate the program or the reasoning and program, depending on the experimental setting.

In this supervision setting, after the model generates its response, we provide the reference answer and prompt the model to learn from this standard. We also provide the model with the requirement to examine the difference between the generated answers and the standard ones and learn from the correct answers. This process involves iterative learning, where the model practices generating programs from all the examples, receiving feedback, and refining its approach before generating the required programs for new claims. The simplified supervised exemplar of Chain-of-thought (CoT) is shown in list 4.4.

```
{
  "role": "system",
  "content": (System instruction: Generate an analysis and a pseudo
program to verify a claim step-by-step using the corresponding
constituency parsing tree, following specific rules for functions
and logical operators.)
},
{
  "role": "user",
  "content": "The claim is that Howard University Hospital and
Providence Hospital are both located in Washington, D.C.\nThe
constituency parsing tree is:
Token      PoS      3      4      5      6      7      8
-----
Howard      PROPN
University PROPN
Hospital    PROPN
and          CCONJ
Providence  PROPN
Hospital    PROPN
are          AUX
both        PRON
located     VERB
in          ADP
Washington PROPN
,           PUNCT
D.C         PROPN
.           PUNCT

```

```

}
{
  "role": "assistant",
  "content": (Generated parsing tree analysis and program code)
},
{
  "role": "user",
  "content": "The suggested analysis is: (Suggested analysis).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},

```

LISTING 4.4: Chain-of-Thought with Supervision prompt exemplar

4.4 Program Execution

4.4.1 Function Handler

In our system, the function handler is a core component of the program execution stage, responsible for processing various functions generated during the program generation phase. According to the function type mentioned in the previous section 2.2.2, which are question, verify, and predict functions, we have designed three main types of corresponding function handlers.

1. **Question Handler:**

The question handler is responsible for processing interrogative sentences. It takes a question Q as input and returns a string answer A . We utilize FLAN-T5 for this task, which has been trained on over 1,800 tasks and has achieved state-of-the-art zero-shot performance on a variety of QA benchmarks. The prompt used by this module is listed below:

```
[EVIDENCE] Q: [QUESTION] ? The answer is:
```

2. **Verify Handler:**

The verify handler functions as a fact verification module. It accepts a claim C as input and returns a label of either TRUE or FALSE. We also employ FLAN-T5 for this module, using the following prompt:

```
[EVIDENCE] Q: Is it true that [CLAIM]? True or False?  
The answer is:
```

3. **Predict Handler:**

The predict handler performs logical operations over variables established in previous steps. It takes as input a logical expression that can include AND and OR operations. The output is the predicted veracity label for the entire claim.

Chapter 5

Experiments

This chapter presents the experimental design and the corresponding results of our study. Section 5.1 shows the setup of our experiment, including the dataset, models, and the design of our studies. Section 5.2 and section 5.3 present the results and initial observations of our experiments. Section 5.4 offers a further analysis based on the result and then outlines the findings of our study.

5.1 Experimental Setup

5.1.1 Dataset

In our approach, we mainly focus on the HOVER (Happy VERification) dataset[93]. It is specifically designed to address complex claims requiring multi-hop reasoning for verification. The dataset comprises a collection of claims sourced from Wikipedia, where each claim’s validation requires synthesizing information from as many as four English articles. This requirement for multi-hop reasoning and multiple data sources is critical, as it mirrors the complex logical structures often encountered in real-world scenarios.

As we mainly focus on improving LLMs’ ability to decompose complex sentences and generate coherent programs, the complexity of the dataset, including structural complexity and multi-hop reasoning, is of great importance. However, commonly used datasets often present simple scenarios where the information required for answering or verifying a claim is readily apparent within a single text snippet or through direct inference. HOTPOTQA, Thus most existing datasets including FEVER and HOTPOTQA are not complex enough to meet the requirements of our approach [94], [95]. In contrast, the HOVER dataset requires the model to engage in a deeper, more intricate process of claim decomposition and verification. It needs the retrieval and integration of information from multiple hops to verify a claim, ensuring the model must perform rigorous reasoning rather than relying on superficial text matching or straightforward inference. This aspect of the dataset is particularly beneficial for our project, as we aim to develop and test a methodology that can efficiently handle complex, multihop claims. This alignment with our research objectives makes the HOVER dataset an ideal choice for exploring advanced verification techniques within a more complex and practical scenario.

The HOVER dataset is a specialized resource designed for fact verification, containing a total of 4,000 data entries. This dataset is balanced with an equal proportion of refuted and supported claims, maintaining a 1:1 ratio. It is characterized by its composition of multi-hop claims, which are pivotal for evaluating the depth and complexity of reasoning required in claim verification. Specifically, the HOVER dataset comprises 80% of 2-hop claims, 39% of 3-hop claims, and 15% of 4-hop claims.

Given the complexity and detailed nature of our experiments, particularly those involving the incorporation of extensive textual information such as parsing trees and decomposition analysis in prompts, evaluating the entire dataset would result in prohibitive costs. To manage these expenses efficiently while maintaining experimental integrity, we opted to extract a subset of the dataset, consisting of 500 entries. This subset was carefully curated to mirror the original dataset’s characteristics, ensuring that the balance of refuted to supported claims remains at 1:1. Additionally, the proportion of n-hop claims in our subset aligns with their distribution in the full dataset, preserving the structural and compositional integrity of the original HOVER dataset. This strategic sampling allows for a cost-effective yet comprehensive evaluation of our models under various experimental settings.

5.1.2 Models

In this experiment, we utilized several large language models for program generation, specifically Mixtral 8x7, Qwen2 72B, and GPT models. We opted not to use the Llama model due to its limited context window, which is insufficient to accommodate all our exemplars during prompting. These models are well-known in the field, and we tested across different scales to derive generalizable conclusions and evaluate the adaptability of our method across various models.

Additionally, we also focused on the cost efficiency of the models used. Table 5.1 shows the pricing of the models used in our experiment. The pricing information for the GPT series models used in our experiments is sourced from the official website data available from May-June 2024 ¹. It’s important to note that these figures may differ from current prices due to potential changes in pricing policies or updates to the models themselves. Although the Mixtral and Qwen models are open-source, our lack of high-performance computing resources necessitated the use of a third-party service to run these models. For this purpose, we utilized the API provided by TogetherAI ², which offers inference services for a variety of models with quick processing times.

Pricing per 1M tokens	Input	Output
Mixtral 8x7b	0.6\$	0.6\$
Qwen2 72b	0.9\$	0.9\$
GPT-3.5-turbo	0.5\$	1.5\$
GPT-4	5\$	15\$

TABLE 5.1: Pricing of Models (in US Dollars)

GPT

GPT-3.5 [6] is part of OpenAI’s Generative Pre-trained Transformer series. Features 175 billion parameters, it is one of the largest language models available. It is trained on a diverse dataset to perform a wide array of language-related tasks, including translation, summarization, and even code generation. Its versatility and capacity for context-aware language generation have made it a benchmark in the field. As the model is trained extensively, GPT3.5 can generate coherent and contextually relevant text across various topics. It has been widely applied in everyday conversational AI to complex problem-solving because of its adaptability and broad utility.

¹<https://openai.com/api/pricing/>

²<https://www.together.ai/>

GPT-4 marks a significant advancement over GPT-3.5, offering improvements in performance, efficiency, and safety. Although specific details about its architecture and parameters are not fully disclosed, GPT-4 is known for its enhanced ability to generate accurate and relevant text. It includes extensive training on up-to-date data, improving its contextual understanding and response quality. The model incorporates advanced fine-tuning techniques and safety mechanisms to minimize biases and ensure ethical use.

Mixtral

Mixtral is designed for domain-specific tasks with high accuracy and interpretability. It is a sparse mixture-of-experts network and a decoder-only model where the feedforward block selects from eight distinct parameter groups [7]. Mixtral matches or exceeds the performance of GPT-3.5 on several benchmarks. It excels in mathematics, code generation, and multilingual tasks, significantly outperforming Llama 2 70B in these areas. Mixtral effectively retrieves information from its context window regardless of sequence length and the location of information. The primary configurations include Mixtral 8x7B and Mixtral 8x22B, each tailored to leverage both rule-based methods and deep learning techniques for precise outputs in specialized domains. Mixtral 8x7B consists of eight interconnected models, each with 7 billion parameters, while Mixtral 8x22B with each model having 22 billion parameters.

Qwen

Qwen, derived from the Chinese phrase "Qianwen" meaning "thousands of prompts," is a comprehensive language model series encompassing various models with different parameter counts [96]. The series includes base pre-trained models, chat models fine-tuned with human alignment techniques like supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF), as well as specialized models for coding and mathematics. The base Qwen models have been extensively trained on up to 3 trillion tokens of diverse texts and codes, demonstrating superior performance across numerous tasks compared to larger counterparts. Qwen-CHAT models are fine-tuned on curated datasets for tasks such as chat, tool use, and safety, showing competitive performance, although slightly behind GPT-4 in benchmarks.

Qwen addresses diverse needs with many specialized models, achieving remarkable results in various domains. Specialized models like CODE-Qwen, which includes CODE-Qwen-7B and CODE-Qwen-14B, excel in code generation, debugging, and interpretation, performing highly on benchmarks such as HumanEval and MBPP. MATH-Qwen-CHAT models, including MATH-Qwen-7B-CHAT and MATH-Qwen-14B-CHAT, are designed for mathematical problem-solving and outperform open-source models of similar sizes, nearing GPT-3.5 performance on math-related benchmarks like GSM8K [97] and MATH [98].

5.1.3 Baseline

Our baseline setting in this experiment involves not incorporating any linguistic features and not providing supervision during prompting. For each model used, we have established a corresponding baseline result. This approach allows for horizontal comparison within each model, enhancing the persuasiveness of our analysis.

5.1.4 Study Design

Our study mainly focuses on a comprehensive experimental framework designed to explore the integration of various parsing trees, reasoning paths, and prompting formats. As we have three distinct types of parsing trees, three reasoning paths, and two prompting formats, ideally there should be a total of eighteen experimental combinations. However, due to constraints in time and resources, we strategically divided our experiments into two primary studies, each focusing on different aspects of the framework. Table 5.2 provides a clear view of the experimental settings for two different studies.

Experimental Setting	Linguistic Feature	Reasoning	Supervision
Study One	Dependency parsing, Constituency parsing, AMR	NR	NS
Study Two	Constituency parsing	NR, CoT, ToT	S, NS

TABLE 5.2: Experimental Setting for Studies

1. Study One: The impact of linguistic features

The first study focused exclusively on the analysis based on different parsing trees. For this part of the study, we employed the Mixtral 8x7 model [7], a robust system capable of handling complex linguistic structures. We conducted these experiments across the entire HOVER dataset, utilizing three different parsing trees to assess how variations in syntactic analysis impact the outcome of our claim verification processes.

In terms of reasoning and prompting, we simplified the experimental settings to a "No Reasoning - No Supervision" (NR-NS) configuration. This setting was chosen to isolate the effect of the parsing trees from other variables, providing a clear view of how different syntactic structures alone can influence the performance of the language models in complex claim verification tasks.

2. Study Two: The impact of reasoning and supervision

The second study was tailored to evaluate the combined effects of reasoning paths and prompting formats. In this setup, we standardized the use of the constituency parsing tree as our linguistic feature across all experiments to maintain consistency in the syntactic analysis while varying the cognitive and interactive aspects of the models. Because of the budget limit, we only used the subset of the HOVER dataset.

We deployed a total of six experimental combinations with three different models: Mixtral 8x7 [7], Qwen2 72B [96], and GPT-3.5 [6]. These models were selected for their varying capabilities and approaches to handling reasoning tasks and interaction formats. Notably, the "Tree of Thought" reasoning method required a more advanced understanding and generation capability, which the aforementioned models struggled with due to their limited ability to execute "single-step generation" commands effectively. Consequently, for experiments utilizing the Tree of Thought method, we opted to use GPT-4, leveraging its enhanced cognitive capabilities to better accommodate the complexity of this specific reasoning approach.

In designing the experimental groups, we prioritized both technical feasibility and effective resource management. This division allowed us to explore a significant portion of the experimental matrix within the constraints of our resources, although it limited the breadth of variable interactions. Additionally, to minimize the influence of randomness and ensure the accuracy of our results, we conducted each experimental setting three times and took the average values as the final result. This replication provides a robust basis for our findings, ensuring that observed effects are consistent and reliable across multiple trials.

5.2 Study One: Impact of Linguistic Features

5.2.1 Main Results

Table 5.3 presents the Macro-F1 score of our experiments conducted across the different parsing trees. The Macro-F1 score is particularly valuable in scenarios where class balance is crucial, as it calculates the average of the F1 scores for each class independently, treating all classes with equal importance regardless of their frequency. This makes it an ideal metric for our study, given the balanced nature of the HOVER dataset between 'refute' and 'support' claims, ensuring that both types of outcomes are equally represented and assessed in the analysis.

	2 hop	3 hop	4 hop	total
Baseline	53.64	52.53	51.88	52.68
Dependency parsing	52.75	52.81	51.68	52.55
Constituency parsing	54.80	50.74	51.59	52.08
AMR 55.06	54.50	52.65	54.26	

TABLE 5.3: Macro-F1 Score for Study One

To assess the statistical significance of the observed differences in F1 scores across various settings, we performed a T-test. A T-test is a statistical test that is used to compare the means of two groups to determine if there is a significant difference between them. It is particularly useful in experiments like ours where the aim is to figure out the effectiveness of different experimental variables on model performance. The result of the two-tailed T-test is shown in table 5.4.

	Dependency Parsing	Constituency Parsing	AMR
p-value	0.95	0.77	0.47

TABLE 5.4: P Values for the Result of Study One

As we focus on the universality of our approach and its adaptability to different fields, cost is also within our evaluation scope. The pricing of different models was mentioned in section 5.1.2. We only present the total cost of running this study on 4000 data entries, which is shown in table 5.5.

	Baseline	Dependency Parsing	Constituency Parsing	AMR
Cost	1.23\$	2.71\$	2.90\$	2.60\$

TABLE 5.5: Cost (in US Dollars) for Study One

5.2.2 Observations

1. Adding linguistic features will not significantly increase costs.

Based on the table 5.5, the inclusion of parsing trees in our experiments did not significantly increase the overall cost. This suggests that integrating these structural components can enhance the model’s comprehension and decomposition abilities without a substantial increase in financial expenditure.

2. Adding linguistic features is not statistically significant.

From the table 5.4, the p-values obtained from the T-test are greater than 0.05. This indicates that the differences in F1 scores between baseline and adding linguistic features are not statistically significant. Further experiment is required to draw general conclusions.

3. AMR enhances semantic understanding and sentence decomposition, while syntactic features show limited impact.

Among all the parsing trees tested, AMR demonstrated the most significant performance improvement. The F1 scores shown in 5.3 indicate that while the results for constituency and dependency trees were nearly identical to the baseline, AMR surpassed the baseline by approximately 2%. Moreover, AMR consistently outperformed the baseline across different hop numbers. These findings suggest that AMR can enhance the model’s semantic understanding to a noticeable extent, making it a valuable asset in parsing complex claim structures.

On the other hand, the performance of the syntactic trees either had no impact on the overall results or even showed a decline. This decline in performance for syntactic trees leads us to speculate that semantic representations, like those provided by AMR, might be more useful because they comprehensively capture all semantic information.

5.3 Study Two: Impact of Reasoning and Supervision

5.3.1 Main Results

Table 5.6 presents the Macro-F1 score of our experiments conducted across the different reasoning methods and supervision settings. Table 5.7 shows the result of the two-tailed T-test. Table 5.8 shows the total cost incurred during the entire experimental process of study two, which only focuses on a subset of 500 data entries.

	Baseline	NR-NS	NR-S	CoT-NS	CoT-S	ToT-NS	ToT-S
Mixtral 8x7B	61.60	60.40	60.60	58.60	55.80	–	–
Qwen2 72B	64.20	63.40	63.00	64.40	60.40	–	–
GPT-3.5	63.00	62.60	61.60	62.80	58.40	–	–
GPT-4	64.80	–	–	–	–	63.20	64.00

TABLE 5.6: Macro-F1 scores for Study Two

As shown in the table 5.7, all the p-values obtained from the T-tests are greater than 0.05. This indicates that the differences in F1 scores between the various experimental settings are not statistically significant. Therefore, we conclude that the

	NR-NS	NR-S	CoT-NS	CoT-S	ToT-NS	ToT-S
Mixtral 8x7B	0.92	0.51	0.80	0.95	–	–
Qwen2 72B	0.59	0.61	0.87	0.71	–	–
GPT-3.5	0.68	0.91	0.85	0.53	–	–
GPT-4	–	–	–	–	0.76	0.57

TABLE 5.7: P Values of the Result of Study Two

Cost	Baseline	NR-NS	NR-S	CoT-NS	CoT-S	ToT-NS	ToT-S
Mixtral 8x7B	1.23\$	2.90\$	6.73\$	10.34\$	14.69\$	–	–
Qwen2 72B	1.85\$	4.32\$	10.05\$	15.48\$	22.00\$	–	–
GPT-3.5	1.10\$	2.50\$	5.69\$	9.10\$	12.87\$	–	–
GPT-4	11.11\$	–	–	–	–	37.39\$	62.64\$

TABLE 5.8: Cost (in US Dollars) for Study Two

variables tested (parsing trees, reasoning paths, and prompting formats) do not have a significant impact on the accuracy of the models as measured by the F1 score.

5.3.2 Observations

Based on the results from our various experimental settings, we observed key trends that influence model performance, particularly concerning different parsing trees and prompting methods. These observations are summarized as follows:

1. Superior Performance of Qwen2 Model.

The Qwen2 72B model has outperformed the Mixtral 8x7B and GPT-3.5 model in this task, and even performs similar to GPT-4. This highlights the advancements in the Qwen model’s capabilities, particularly in handling complex language processing tasks. We assume this is due to the fact that Qwen is trained better on sentence decomposing task, and also it is the newest version [96].

2. Suitability of Models with Lower Input Token Costs

Given our experimental settings, which resulted in a much higher number of input tokens compared to output tokens, models with lower input token pricing are more suitable for this task. This cost-efficiency is particularly relevant when dealing with large datasets where the volume of input data can dramatically influence total costs.

3. Advanced prompting methods have a negative impact on decomposition.

Contrary to our hypothesis **H2** that reasoning and supervision can facilitate LLMs’ ability to decompose sentences, our results show otherwise. The baseline F1 scores were consistently higher than those of other settings, with only one exception. This trend was particularly obvious in the lightweight Mixtral 8x7b model, where the worst-performing method lagged behind the baseline by as much as 6%. Among all the settings, the Chain of Thought with Supervision (CoT-S) method exhibited the poorest performance, consistently scoring at least 4% lower than the baseline. This indicates that while theoretically promising, the practical application of these advanced prompting methods may require refinement to avoid detrimental effects.

5.4 Further Analysis on Study Two

Given the unexpected outcomes related to the reasoning and supervision prompting methods, which did not align with our initial hypotheses and expectations, a more detailed approach is required. To address these issues, we have decided to perform a series of case studies specifically targeting these aspects of our experimental framework.

The primary motivation for employing case studies in this context is to gain deeper insights into the nuances of how different prompting methods influence the generation of executable programs. Due to the complexity of the programs generated and the current lack of appropriate evaluation methodologies that accurately measure the effectiveness of such programs, traditional metrics like the F1 score are insufficient for our needs. Therefore, in these case studies, we will shift our evaluation strategy from relying on quantitative metrics such as the F1 score derived from program execution outcomes, to a qualitative assessment conducted by manual review. This approach involves a detailed examination of the programs generated by the models under various prompting conditions. Reviewers will assess whether these programs logically and effectively represent the original claims' information and meaning. The criteria for evaluation will focus on the program's coherence, logical structure, and fidelity to the original claim.

5.4.1 Program Analysis

Before initiating our case studies, it is essential to establish clear and effective evaluation criteria for the programs generated by our models. Our focus will be on highly contentious claims, where the programs exhibit a minimal difference or even equality between support and refute outcomes. This is because these claims are more likely to contain a diverse array of logical flaws or misrepresentations, making them ideal candidates for a detailed evaluation.

We selected ten claims that are deemed the most contentious for each hop level. After analyzing 540 programs (18 programs per claim \times 10 claims per hop level \times 3 hop levels), we have broadly categorized occurred errors into three main types: format errors that prevent normal execution, errors occurring during the decomposition of claims, and errors arising from the conversion of subclauses into the program. In this section, we will demonstrate the various types of errors that may occur in the generated program in detail and attempt to analyze the causes of these errors.

Format Errors

Format errors occur when the structure of the generated program does not conform to the format specified and exemplified in our setting. Because the structure cannot be correctly parsed, these programs fail to execute properly. When we encounter such cases, we classify the execution results of these programs as errors. This means that the outcomes of these programs are neither "refute" nor "support," and they will never match the standard answers. Such programs can never be considered correct in terms of execution results.

Format errors can be further subdivided into three categories:

1. **Context Retention Failure**

This type of failure occurs when the program fails to correctly retain the variables used in previous steps. These programs may reference a variable that

has never been defined or declared, leading to execution failure. Here is an example of programs with context retention failure:

```
fact_1 = Verify(subclause_1)
fact_2 = Verify(subclause_2)
label_1 = Predict(f_1 and f_2)
```

From the first two `Verify()` functions, we can observe the variables used are `fact_1` and `fact_2`. However, the model fails to accurately remember the previously generated variable names, so in the final step, it uses `f_1` instead of `fact_1`.

2. Structural Errors

Structural errors occur when the generated program uses syntax structures that we have not specified. Some programs may use advanced programming syntax structures, such as conditional statements or loop statements. Others may generate more than one `Predict()` statement at the end of the program or fail to generate a `Predict()` statement at all. Here is an actual generated program with structural error:

```
fact_1 = Verify("The film The Wild was released.")
answer_1 = Question("Which film was released first, The
Country Bears or The Wild?")
if answer_1 == "The Wild":
    label = Predict(True)
else:
    label = Predict(False)
```

3. Function Misuse

Function misuse errors arise when the program does not correctly use the functions we have specified. These errors include treating the return value of a fact function as a variable within a claim or question, using unsupported logical operators (such as `!=` or `xor`) within the predict function, performing verification within the predict function, or including extraneous parentheses in the predict function. According to our specifications, the predict function is only meant to integrate the results of various steps, and the only supported logical operators are `and` or `or`. Here is an example of function misuse:

```
fact_1 = Verify(sub_claim1)
fact_2 = Verify(sub_claim2)
fact_3 = Verify({fact_1} publishes {fact_2}.)
label = Predict(answer_1 == answer_2)
```

Decomposition Errors

Decomposition errors occur when the model makes mistakes while breaking down a claim into subclauses. Due to the experimental setup, we can clearly see the subclauses resulting from claim decomposition only in the chain of thought method. In the other two experimental setups, we can only infer the subclauses from the content

of the generated program. These errors are critical because the quality of the decomposition directly impacts the accuracy and reliability of the final program execution.

Decomposition errors can be broadly classified into five categories:

1. Insufficient Decomposition

This issue arises when the model fails to thoroughly break down all the subclauses, leaving one or more subclauses still containing two or even more hops. In the most extreme cases, the subclause remains identical to the original claim without any decomposition. Insufficient decomposition can lead to overly complex subclauses that the executor cannot properly understand or run, significantly impacting the programs performance and accuracy. Here is an example of a program with insufficient decomposition. The program fails to return the correct answer due to the high complexity of the subclause that returns fact_1.

```
# The claim is that The attraction at universal studios that was
   based on "The Tonight Show" replaced an attraction that
   replaced an attraction based on the movie Fear Factor.

answer_1 = Question("What is the attraction at universal studios
   that was based on 'The Tonight Show'?")
fact_1 = Verify("The attraction {answer_1} replaced an attraction
   that replaced an attraction based on the movie Fear Factor.")
label = Predict(fact_1)
```

2. Over Decomposition

Contrary to intuition, more decomposition is not always better. Completely breaking down a claim into several simple sentences can sometimes alter the original meaning or result in the loss of crucial information. Here is an example of over-decomposition:

```
# The claim is that David Luchins served as an advisor to a Vice
   President who passed away in 1978.

answer_1 = Question("Who did David Luchins serve as an advisor to
   ?")
fact_1 = Verify("{answer_1} was a Vice President.")
fact_2 = Verify("{answer_1} passed away in 1978.")
label = Predict(fact_1 and fact_2)
```

This decomposition seems to be correct, but it misses one key constraint, leading to the wrong answer. The Vice President mentioned in the claim who passed away in 1978 is Hubert Horatio Humphrey Jr. David Luchins did serve as an advisor to Hubert Horatio Humphrey, but not during his tenure as Vice President. Thus the correct decomposition should be:

```
answer_1 = Question("Who is the Vice President that David Luchins
   served as an advisor to?")
fact_1 = Verify("{answer_1} passed away in 1978.")
label = Predict(fact_1)
```

3. Information Missing

This error occurs when the subclauses fail to cover all the information present in the original claim. This type of error can also occur during the conversion process, where the subclauses are correctly decomposed, but some information is lost when converting them into the program. This error occurs when the subclauses fail to cover all the information present in the original claim. Certain qualifiers and adjectives are easily overlooked by the model. Such errors do not necessarily lead to incorrect execution results because the missing information might not be decisive for the overall semantics. However, they still represent a failure in accurately capturing the full scope of the original claim. As the example shown below, during the decomposition process, "(I'm 10,000 Years Old)" and iconic rock and roll singer are omitted.

```
# The claim is that "(I'm 10,000 Years Old)" was the subtitle of
  the 11th album by the iconic rock and roll singer, Elvis
  Presley.

answer_1 = Question("What was the subtitle of the 11th album by
  Elvis Presley?")
fact_1 = Verify("{answer_1} was the subtitle of the 11th album.")
label = Predict(fact_1)
```

4. Adding External Information

LLMs are trained on vast amounts of data, which inherently includes some external data. During the decomposition process, the model might unconsciously identify and incorporate information related to the claim from its training data into the subclauses. Although this external information is often highly accurate, it is considered an error in our context because we focus solely on the decomposition process. The inclusion of external information can lead to deviations from the original claim, impacting the validity of the decomposition. For example, the program shown below mentions a name, Spike Lee, that is not in the original claim. Even though Spike Lee is the director of "She Hate Me" and owns the production company 40 Acres and the Mule Filmworks, it is still considered an error. This is because we mainly focus on the preciseness of decomposition, but not the accuracy of the execution result.

```
# The claim is that The production company 40 Acres and a Mule
  Filmworks is owned by the director of "She Hate Me".

fact_1 = Verify("40 Acres and a Mule Filmworks is owned by Spike
  Lee.")
fact_2 = Verify("Spike Lee is the director of She Hate Me.")
label = Predict(fact_1 and fact_2)
```

5. Semantic Unintelligibility

These errors may stem from the model's inability to fully understand the claim's semantics or specific terms within it. This leads to errors in the decomposed subclauses, including but not limited to reversing the intended meaning or misidentifying the referents. Semantic errors are particularly problematic as they can lead to fundamentally incorrect interpretations and subsequent program execution results. Here is an example of a semantic unintelligibility error in that the meaning of the program is the opposite of the meaning of the original claim.

```
# The claim is that American singer and songwriter Tori Amos has
  a mezzo-soprano vocal range, where the singer who released The
  Stories Are True doesn't.

answer_1 = Question("Which singer has a mezzo-soprano vocal range
  , where the singer who released The Stories Are True doesn't
  ?")
fact_1 = Verify("The singer who released The Stories Are True has
  a mezzo-soprano vocal range.")
fact_2 = Verify("Tori Amos doesn't have a mezzo-soprano vocal
  range.")
label = Predict(fact_1 and fact_2)
```

Conversion Error

Conversion errors occur when the model correctly decomposes a claim into subclauses but introduces mistakes when converting these subclauses into a program. Similar to decomposition error, we can clearly observe this type of error only in the chain of thought setting, where the model generates steps showing the decomposition of subclauses. In the other two settings, while the steps are not explicitly shown, we can still identify these errors in the generated program.

Conversion errors can be categorized into four types:

1. Over-Complexification

When dealing with a subclause of a statement, the simplest conversion method is to directly use the `verify` function to check the subclause. However, sometimes the model complicates the process by first using the `question` function to ask about a specific component of the subclause with special interrogative words, then using the obtained component in a subsequent `verify` function. Although the generated program appears semantically correct, this unnecessary complexity increases the likelihood of errors occurring during execution in these convoluted parts. For example, the `Question()` and `Verify()` functions shown in the example below are highly semantically repetitive. This combination can be simplified into a single `Verify()` function call.

```
# The claim is that "(I'm 10,000 Years Old)" was the subtitle of
  the 11th album by the iconic rock and roll singer, Elvis
  Presley.
```

```
answer_1 = Question("What was the subtitle of the 11th album by
    Elvis Presley?")
fact_1 = Verify("{answer_1} was the subtitle of the 11th album.")
label = Predict(fact_1)
```

2. Semantic Discontinuity

This issue often arises when the model uses the question function to obtain a critical component of the claim but fails to use this component in the subsequent verify function. This type of error tends to occur with semantic ambiguity error. The lack of continuity between the obtained information and its application results in a disjointed semantic flow. As the example shown below, the variable `answer_1` has never been used in the following function call, even though this variable covers the key information in the original claim.

```
# The claim is that Simon Helberg is the American actor, writer,
    and musician, born in 1980, who appeared with Jason Ritter in
    "Ode to Janice Melt".
```

```
answer_1 = Question("Who is the American actor, writer and
    musician, born in 1980?")
fact_1 = Verify("Simon Helberg appeared with Jason Ritter in 'Ode
    to Janice Melt'.")
label = Predict(fact_1)
```

3. Semantic Ambiguity

Such errors typically occur when pronouns are not properly handled during the conversion of subclauses into a program. In a passage, each sentence is closely linked with the preceding and following sentences, allowing the model to clearly understand what a pronoun like "the" refers to. However, when converting to a program, each function operates independently without any context from other functions. This lack of context means the model cannot determine what "the" refers to within a single function, leading to semantic ambiguity.

```
# The claim is that Zamalek Sporting Club Centennial was a
    friendly match against professional football club that plays
    in La Liga.
```

```
fact_1 = Verify("Zamalek Sporting Club Centennial was a friendly
    match.")
fact_2 = Verify("The match was against a professional football
    club.")
fact_3 = Verify("The professional football club plays in La Liga
    .")
label = Predict(fact_1 and fact_2 and fact_3)
```

4. Overly Broad Questioning

During the conversion process, the model might need to choose the correct subclause from multiple options and use its information for filtering. Thus choosing the best subclause is of great importance for the quality of generating the program. However, some subclauses may provide information that is too common or general, which lacks sufficient constraint. This means the information within the subclause is too vague to accurately filter out a unique answer.

```
# The claim is that Zamalek Sporting Club Centennial was a
    friendly match against professional football club that plays
    in La Liga.

answer_1 = Question("Which professional football club plays in La
    Liga?")
fact_1 = Verify("Zamalek Sporting Club Centennial was a friendly
    match against {answer_1}.")
label = Predict(fact_1)
```

5.4.2 Case Study

To delve deeper into the qualitative aspects of the programs generated by our models and to understand why the baseline setting consistently outperforms others, we are initiating targeted case studies. These studies will focus on two main objectives: assessing the nuanced quality of the programs beyond mere correctness and uncovering the reasons behind the baseline's superior performance. Each study analyzes 20 claims, specifically examining the programs generated by the Qwen model. The Qwen model was selected for these studies due to its recency and sophistication, which are representative of current models. It also allows us to derive valuable insights that could inform the development of future high-performance models.

Why Baseline Outperforms Other Settings?

In our exploration of why the baseline setting consistently exceeds the performance of other experimental settings, we conducted a focused case study involving a careful selection of 20 claims from the subset of 500 claims. We aim to delve deeper into the quality and integrity of the programs generated under the baseline setting compared to others. This involves assessing whether the baseline programs are genuinely superior in terms of their semantic accuracy and logical consistency, or if they merely yield correct results by coincidence.

The selection process was designed to maintain the proportion of hops consistent with the overall distribution in the HOVER dataset. A total of 20 claims were selected based on the specific criteria below that highlight the comparative effectiveness of the baseline setting. The claims were chosen according to the following principles:

1. Baseline Accuracy

All selected claims must have correct execution results under the baseline setting. This ensures that the baseline is performing effectively for these claims.

2. Low Accuracy in Other Settings

The claims selected generally exhibit poor performance in settings other than the baseline. Ideally, these are claims where the baseline is the only setting that yields a correct result, or where very few other settings manage to produce correct results.

What is the Distribution of Different Errors in Programs?

This case study aims to analyze the distribution of different types of errors in the programs generated by our models for decomposing controversial claims. We focus on how well these programs represent the original claims' information and logic, and how linguistic features and different prompting methods influence the models' effectiveness in decomposition.

To conduct this analysis, we selected the 20 most controversial claims from the subset of 500 claims extracted from the original HOVER dataset, specifically choosing those with the most balanced support and refute outcomes. While ensuring these claims were selected, we maintained the original dataset's proportionality in terms of hop numbers and claim outcomes. This approach guarantees that our analysis is representative and unbiased, offering a balanced view of the model's capabilities across varied claim complexities. By examining these controversial claims, we can better understand the common errors in program generation and their respective proportions, providing valuable insights for future improvements.

Results and Observations

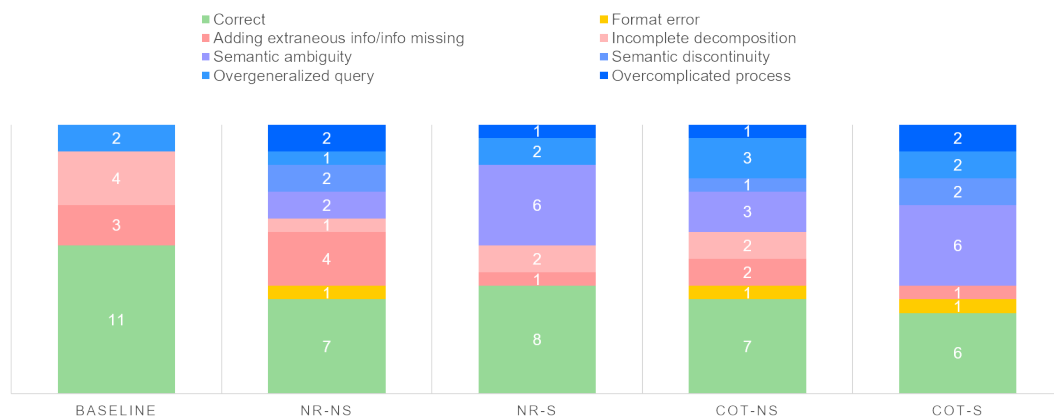


FIGURE 5.1: Distribution in Program Generation for 'Why Baseline Outperforms Other Settings'

Figure 5.1 and figure 5.2 shows two stacked bar charts representing the result of two case studies. To aid in the visualization and interpretation of our results, we adopted a uniform color scheme across the graphical representations in both studies. This approach not only enhances the clarity of our findings but also facilitates a straightforward comparison between different types of errors within and across the case studies. In our stacked bar charts, the following color coding is used:

1. **Green:** correct programs, which means properly decomposes and accurately represents the original claim's semantics, regardless of the program's execution outcome.

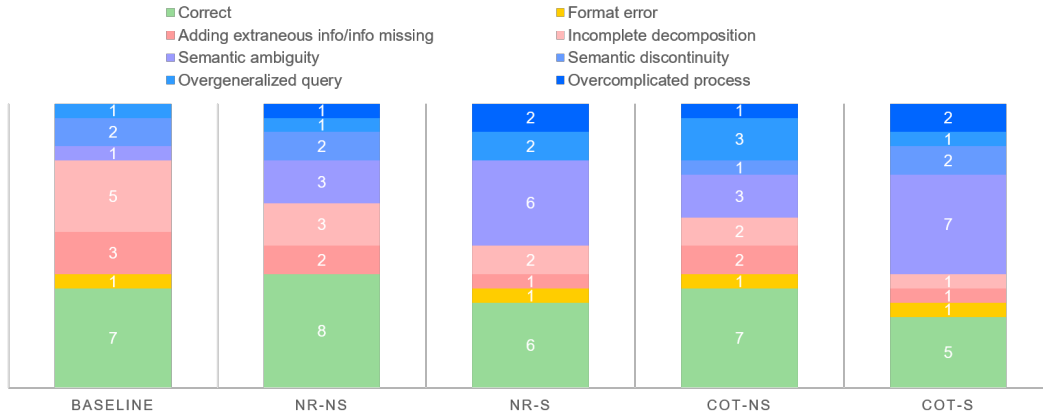


FIGURE 5.2: Error Distribution in Program Generation for 'What is the Distribution of Different Errors in Programs?'

2. **Yellow:** format errors.
3. **Red:** decomposition errors, including adding/missing information and incomplete decomposition.
4. **Blue and Purple:** conversion errors, including semantic ambiguity, semantic discontinuity, overgeneralized query, and overcomplicated process.

Here are the observations from the result:

1. **Correct execution result may not necessarily guarantee accurate program logic.**

From the case study 'Why Baseline Outperforms Other Settings', which is shown in figure 5.1, despite all 20 claims being selected based on their correct execution outcomes in the baseline setting, only 11 of these programs (55%) were actually correct in their logical composition and execution. However, in other settings with generally incorrect execution results, each still had at least 6 programs (30%) that were accurate. This reveals that correct outcomes do not always reflect accurate program logic, and model logic can be correct even when execution outcomes suggest otherwise.

2. **Parsing trees and varied prompting methods can enhance decomposition but may increase conversion errors.**

Baseline settings tend to have a higher ratio of decomposition error compared to other settings. From the first case study shown in figure 5.1, the baseline setting exhibited the highest rate of decomposition errors at 78% (7 out of 9), while other settings reduced errors to a maximum of 38% (5 out of 13). Moreover, the COT-S setting particularly stood out, with decomposition errors at just 8% (1 out of 12), highlighting the substantial improvement prompted methods can offer. This demonstrates the effectiveness of integrating syntax trees and varying prompting methods in improving decomposition. We can also witness the same patterns in the second case study in figure 5.2. While 61% (8 out of 13) of the false programs are due to decomposition errors in the baseline setting, the highest rate among other settings is only 42% (5 out of 12).

On the other hand, parsing trees and varied prompting methods may increase conversion errors, as they tend to have a larger portion among the errors. From figure 5.1, in the baseline, conversion errors accounted for only 22% (4 out of 13) of all errors, whereas in other settings, they comprised at least 54% (7 out of 13) of errors, peaking at 86%(12 out of 14). From figure 5.2, there are only 31%(4 out of 13) errors due to conversion in the baseline setting, while there are at least 58% (7 out of 12) errors for other settings. This suggests that while these additions aid in decomposition, they may complicate the conversion process, leading to more errors of this type.

3. Supervised settings increase conversion errors, especially semantic ambiguity.

Supervised settings, especially CoT-S, showed a higher prevalence of conversion errors. In figure 5.1, such error accounts for 86% (12 out of 14) of errors in this category, with No Reasoning with Supervision (NR-S) following at 62% (8 out of 13). Both settings also showed a high incidence of semantic ambiguity errors, which are 50%(6 out of 12) and 43%(6 out of 14). Similarly, in figure 5.2, 80% (12 out of 15) of the errors in setting CoT-S are due to conversions. This observation suggests that while supervision improves certain processing aspects, it also complicates semantic accuracy, increasing conversion errors.

Chapter 6

Discussion

6.1 Answer to Research Questions

6.1.1 Can the addition of linguistic features enhance the decomposition capabilities of LLMs?

Our research suggests that the addition of linguistic features can potentially enhance the decomposition capabilities of LLMs, but the effects are nuanced and context-dependent. While we observed some positive impacts, particularly in reducing decomposition errors during program generation, the overall results were mixed and lacked statistical significance, indicating that the benefits may not be universally applicable or consistently demonstrable across all scenarios.

Our first set of experiments, exploring different parsing trees, showed promising results. The incorporation of AMR led to improved performance in decomposition tasks, indicating that semantically rich representations can aid LLMs in processing complex claims. However, our second set of experiments, using constituency trees with various prompting methods, revealed a more complicated picture. While overall execution accuracy did not improve and even slightly declined, our case study analysis showed a reduction in decomposition errors when linguistic features were used. This apparent inconsistency between macro-level metrics and micro-level improvements highlights the multifaceted nature of enhancing LLM capabilities.

However, the addition of linguistic features led to a decrease in decomposition errors but simultaneously introduced more errors in converting sub-clauses into programs. This explains why we didn't see an improvement in overall execution accuracy despite the reduction in decomposition errors. It suggests that while linguistic features enhance the LLM's ability to break down complex claims, they may also introduce challenges in subsequent processing stages.

It is also important to note that our statistical analysis did not reveal significant differences, indicating that these findings may not be universally applicable. This lack of statistical significance underscores the need for larger-scale studies, more diverse datasets, or alternative statistical approaches to provide more definitive insights.

6.1.2 How effective are different prompting methods in aiding LLMs reasoning during the decomposition process?

This question does not yield a straightforward answer based on our experimental results. Our findings suggest that the effectiveness of different prompting methods varies across different aspects of the decomposition process, with each method presenting its own set of advantages and challenges.

Examining the execution results through F1 scores, we were unable to draw a unified conclusion about the superiority of any single prompting method. Our case studies provided more nuanced insights. We observed that the CoT settings (CoT-NS, CoT-S) performed slightly better in decomposition compared to settings without explicit reasoning prompts. This was evidenced by a marginally lower proportion of decomposition errors, suggesting that encouraging step-by-step reasoning can somewhat improve the LLM's ability to break down complex claims. More notably, settings with supervision showed a marked decrease in decomposition errors compared to those without supervision. This observation leads us to hypothesize that using supervision to enable iterative learning allows the model to better grasp how to correctly utilize provided information, resulting in improved decomposition capabilities. The iterative nature of supervised learning appears to enhance the model's understanding of the decomposition task over time.

However, this improvement in decomposition came with a trade-off. Supervised settings also led to an increase in conversion errors, particularly in semantic ambiguity. We speculate that this setting may inadvertently hinder the model's ability to treat each function in the generated program as independent. This was evidenced by a higher occurrence of pronouns in critical positions such as subjects or objects within the generated programs, potentially leading to ambiguity and errors in the final output.

On the other hand, from a cost-effectiveness perspective, both supervision and CoT settings present less efficiency. The addition of reasoning steps and the implementation of iterative learning significantly increase the input prompt size, leading to higher computational costs and reduced economic efficiency. This practical consideration is crucial when evaluating the overall viability of these prompting methods in real-world applications.

It also should be noted that given that our experimental results did not demonstrate statistical significance after conducting a t-test, coupled with the fact that our case study was limited to analyzing only 20 claims for each selected model program, the findings of this study may lack generalizability. Therefore, the conclusions presented should be considered preliminary and interpreted with caution, serving as a reference rather than definitive evidence.

6.2 Limitation

6.2.1 Experimental Limitation

Limited Data Scope

The conclusions drawn from our experiments may not universally apply due to the limited scope of data tested. Primarily, the experiments were restricted to a subset of one single dataset because of the limited time and budget of calling models' APIs. This restriction narrows the breadth of our conclusions as the dataset may not represent the variability and complexity found in wider applications. The limited dataset scope also raises concerns about the robustness of our findings when applied to different or more extensive datasets, which might exhibit different characteristics and behaviors under similar experimental conditions.

Reduced Experimental Setting

Our innovative approach, which integrates multiple syntax trees, reasoning methods, and prompting formats (with or without supervision), ideally requires a broad set of experimental setups to thoroughly explore all possible interactions and effects. We need to conduct experiments across eighteen different settings based on the combinations of three syntax trees, three reasoning methods, and two prompting formats. However, due to practical limitations, we consolidated these into only two sets of combinations. The first combination involved various pairings of reasoning methods and prompting formats, while the second was an isolated investigation of the syntax trees. This grouping likely resulted in a failure to capture all potential interactions and effects, thereby narrowing the scope of our insights. The limited exploration of these variables can lead to conclusions that do not fully account for the effects of different integrations, potentially missing out on optimal configurations that could enhance performance.

6.2.2 Technical Limitation

Introducing Extra Errors

Our use of a program-based approach inadvertently introduced additional errors. While the model accurately parsed sentences, errors sometimes emerged during the transformation of these sentences into executable programs. This issue illustrates a critical flaw in the program-based approach, where the model's accuracy in sentence parsing does not necessarily translate to error-free program generation. Such discrepancies can alter the outcome of claim verification, affecting the reliability of the results.

Ineffective Use of Parsing Tree Information

Based on our observation and analysis, the effectiveness of using parsing tree information through few-shot prompting has proven to be less than satisfactory. This limitation suggests that our approach might require modifications or enhancements to better leverage structured syntactic data, which is crucial for understanding and processing complex claims.

Dependency on Model and Data

Another considerable technical limitation is the dependency of our methodology on the underlying capabilities of the language model and the quality of the input data. The performance of our approach is highly relevant to these factors, and any shortcomings in the model's understanding of complex linguistic structures or the dataset's integrity directly undermine the effectiveness of the generated programs. This dependency highlights the need for robust model training and comprehensive, high-quality datasets to ensure reliable and accurate verification of complex claims.

6.3 Further Improvement

In this study, we explored the effectiveness of integrating linguistic features, reasoning methods and supervision in prompting to enhance the decomposition capabilities of LLMs in processing complex queries. Our findings indicated that while linguistic features such as semantic parsing provide useful insights, they alone are

not sufficient to significantly enhance decomposition capabilities. Advanced reasoning techniques in prompting and supervisions showed potential in improving the models' ability to handle complex sentence structures through step-by-step reasoning. However, the overall effectiveness varied, suggesting a need for more refined prompting strategies that can dynamically adapt to the complexity of the query.

6.3.1 Experimental Improvements

To address the limitations posed by the restricted experimental settings and data scope, a broader range of experiments should be conducted. This would involve testing additional combinations of experimental settings, including a wider array of linguistic features, reasoning methods, and prompting formats. Expanding the diversity of models and datasets utilized in these experiments would also help in assessing the robustness of our findings across different linguistic and logical structures.

6.3.2 Technical Improvements

While parsing trees have been beneficial, their application could be optimized further. Future work could concentrate on refining the integration of linguistic features by training custom models that inherently utilize these features in their core reasoning processes, rather than merely as additional inputs. Such integration could lead to a deeper understanding and more robust handling of complex linguistic structures. Moreover, investigating alternative methods of leveraging parsing trees to enhance the efficiency of decomposition could prove to be beneficial. For example, experimenting with dynamic tree structures that adapt based on the complexity and specific requirements of each claim could provide more tailored and effective parsing solutions. Additionally, exploring different tree traversal techniques that might better align with the logical flow required for certain types of claims could also improve decomposition accuracy.

Methodologically, creating adaptive prompting techniques that tailor the level of guidance based on real-time model performance would not only refine the models' reasoning capabilities but also their ability to self-correct and learn from iterative feedback. An additional crucial area for future research involves addressing and reducing semantic ambiguities introduced by supervision methods. Our experiments highlighted that while supervision aids in structuring the decomposition process, it can inadvertently introduce errors, particularly semantic ambiguities. Future methodologies should focus on developing supervision techniques that minimize these ambiguities. This could involve the use of advanced semantic analysis tools to ensure that the supervisory feedback aligns closely with the intended meanings and contexts of the processed texts. By refining these supervision methods, we can enhance the precision of the LLMs in understanding and processing complex queries, thus mitigating the risk of semantic inaccuracies.

Chapter 7

Conclusion

This thesis investigated the integration of linguistic features and advanced prompting methods, including reasoning and supervision settings, to enhance the decomposition capabilities of LLMs for processing complex sentences. Our findings showed that incorporating linguistic features has the potential to improve LLMs' understanding of complex sentence structures, providing essential scaffolding for interpreting syntactic and semantic layers effectively.

The study also evaluated various reasoning methods and supervision settings. Structured reasoning paths along with different levels of supervision show promise towards enhancing the model's ability to accurately decompose and analyze complex sentences. On the other hand, while supervision aids in structuring the decomposition process, it also tends to introduce more errors during the conversion from sub-sentences to the program, particularly in terms of semantic ambiguity. This indicates a critical area for improvement in how supervision is implemented. Optimizing how these settings control the flow and integrity of data conversion could mitigate these errors, potentially leading to more reliable and accurate output from the models.

Despite these findings, the study identified several error types in the programs generated by LLMs, including format, decomposition, and conversion errors. These errors highlight the need for further refinement in model training and prompt design.

In conclusion, the insights gained from integrating linguistic features, reasoning, and supervision in prompting positively contribute to advancing LLM capabilities. This thesis lays a foundation for future research aimed at refining AI systems for more sophisticated, accurate, and context-aware processing of human language, encouraging further exploration into optimizing supervision techniques and prompt designs to reduce errors and improve reliability.

Bibliography

- [1] L. E. Nicholas and R. H. Brookshire, "Syntactic simplification and context: Effects on sentence comprehension by aphasic adults," in *Clinical Aphasiology: Proceedings of the Conference 1983*, BRK Publishers, 1983, pp. 166–172.
- [2] J. Lin, "The role of information retrieval in answering complex questions," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, 2006, pp. 523–530.
- [3] J. Liu, "Deconstructing search tasks in interactive information retrieval: A systematic review of task dimensions and predictors," *Information Processing & Management*, vol. 58, no. 3, p. 102522, 2021.
- [4] M. Johnson, "Pcfg models of linguistic tree representations," *Computational Linguistics*, vol. 24, no. 4, pp. 613–632, 1998.
- [5] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer, "A large-scale classification of english verbs," *Language Resources and Evaluation*, vol. 42, pp. 21–40, 2008.
- [6] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [7] A. Q. Jiang, A. Sablayrolles, A. Roux, *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.
- [8] C. Raffel, N. Shazeer, A. Roberts, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.
- [9] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.
- [10] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.
- [11] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27730–27744, 2022.
- [12] T. Alexopoulou, M. Michel, A. Murakami, and D. Meurers, "Task effects on linguistic complexity and accuracy: A large-scale learner corpus analysis employing natural language processing techniques," *Language Learning*, vol. 67, no. S1, pp. 180–208, 2017.
- [13] D. Biber and B. Gray, *Grammatical complexity in academic English: Linguistic change in writing*. Cambridge University Press, 2016.
- [14] X. Lu, "Automatic analysis of syntactic complexity in second language writing," *International journal of corpus linguistics*, vol. 15, no. 4, pp. 474–496, 2010.
- [15] L. Ortega, "Syntactic complexity measures and their relationship to l2 proficiency: A research synthesis of college-level l2 writing," *Applied linguistics*, vol. 24, no. 4, pp. 492–518, 2003.

- [16] C. Shen, J. Guo, P. Shi, S. Qu, and J. Tian, "A corpus-based comparison of syntactic complexity in academic writing of L1 and L2 English students across years and disciplines," *Plos one*, vol. 18, no. 10, e0292688, 2023.
- [17] S. Min, E. Wallace, S. Singh, M. Gardner, H. Hajishirzi, and L. Zettlemoyer, "Compositional questions do not necessitate multi-hop reasoning," *arXiv preprint arXiv:1906.02900*, 2019.
- [18] Q. Chen, "Semantic sentence composition reasoning for multi-hop question answering," *arXiv preprint arXiv:2203.00160*, 2022.
- [19] J. J. Tithi, F. Checoni, and F. Petrini, "Efficient parallel multi-hop reasoning: A scalable approach for knowledge graph analysis," *arXiv preprint arXiv:2406.07727*, 2024.
- [20] L. Pan, X. Wu, X. Lu, *et al.*, "Fact-checking complex claims with program-guided reasoning," *arXiv preprint arXiv:2305.12744*, 2023.
- [21] C. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.
- [22] C. Parsing, "Speech and language processing," *Power Point Slides*, 2009.
- [23] J. Nivre, "Dependency grammar and dependency parsing," *MSI report*, vol. 5133, no. 1959, pp. 1–32, 2005.
- [24] J. Nivre, "Dependency parsing," *Language and Linguistics Compass*, vol. 4, no. 3, pp. 138–152, 2010.
- [25] P. Liang, M. I. Jordan, and D. Klein, "Learning dependency-based compositional semantics," *Computational Linguistics*, vol. 39, no. 2, pp. 389–446, 2013.
- [26] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *Transactions of the association for computational linguistics*, vol. 1, pp. 49–62, 2013.
- [27] L. Banarescu, C. Bonial, S. Cai, *et al.*, "Abstract meaning representation for sembanking," in *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, 2013, pp. 178–186.
- [28] M. Damonte, S. B. Cohen, and G. Satta, "An incremental parser for abstract meaning representation," *arXiv preprint arXiv:1608.06111*, 2016.
- [29] J. Flanigan, S. Thomson, J. G. Carbonell, C. Dyer, and N. A. Smith, "A discriminative graph-based parser for the abstract meaning representation," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 1426–1436.
- [30] Y. Chang, X. Wang, J. Wang, *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.
- [31] M. Ahn, A. Brohan, N. Brown, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.
- [32] Y. Li, X. Ma, S. Lu, K. Lee, X. Liu, and C. Guo, "Mend: Meta demonstration distillation for efficient and effective in-context learning," *arXiv preprint arXiv:2403.06914*, 2024.
- [33] Y.-T. Sung, J.-L. Chen, J.-H. Cha, H.-C. Tseng, T.-H. Chang, and K.-E. Chang, "Constructing and validating readability models: The method of integrating multilevel linguistic features with machine learning," *Behavior research methods*, vol. 47, pp. 340–354, 2015.

- [34] A. Chiche and B. Yitagesu, "Part of speech tagging: A systematic review of deep learning and machine learning approaches," *Journal of Big Data*, vol. 9, no. 1, p. 10, 2022.
- [35] S. Kübler, R. McDonald, and J. Nivre, "Dependency parsing," in *Dependency parsing*, Springer, 2009, pp. 11–20.
- [36] D. Jurafsky and H. Martin, *Jh (2000) speech and language processing*, 2012.
- [37] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1373–1378.
- [38] T. Dozat and C. D. Manning, "Deep biaffine attention for neural dependency parsing," *arXiv preprint arXiv:1611.01734*, 2016.
- [39] M. S. Palmer, D. Gildea, and N. Xue, *Semantic role labeling*. Morgan & Claypool Publishers, 2010, vol. 6.
- [40] L. He, K. Lee, M. Lewis, and L. Zettlemoyer, "Deep semantic role labeling: What works and what's next," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 473–483.
- [41] T. J. Sanders, W. P. Spoooren, and L. G. Noordman, "Toward a taxonomy of coherence relations," *Discourse processes*, vol. 15, no. 1, pp. 1–35, 1992.
- [42] D. Schiffrin, *Discourse markers*. Cambridge University Press, 1987.
- [43] A. L. E. M. L. Robaldo, A. J. R. Prasad, N. Dinesh, and B. Webber, "The penn discourse treebank 2.0," in *Proceedings of the sixth international conference on language resources and evaluation (lrec08), marrakech, morocco, may. european language resources association (elra)*. <http://www.lrec-conf.org/proceedings/lrec2008>, 2008.
- [44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.
- [45] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [46] X. Zhang and M. Lapata, "Sentence simplification with deep reinforcement learning," *arXiv preprint arXiv:1703.10931*, 2017.
- [47] T. Vu, B. Hu, T. Munkhdalai, and H. Yu, "Sentence simplification with memory-augmented neural networks," *arXiv preprint arXiv:1804.07445*, 2018.
- [48] M. Chen, J. Du, R. Pasunuru, *et al.*, "Improving in-context few-shot learning via self-supervised training," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds., Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 3558–3573. DOI: [10.18653/v1/2022.naacl-main.260](https://doi.org/10.18653/v1/2022.naacl-main.260). [Online]. Available: <https://aclanthology.org/2022.naacl-main.260>.
- [49] Y. Gu, L. Dong, F. Wei, and M. Huang, "Pre-training to learn in context," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4849–4870. DOI: [10.18653/v1/2023.acl-long.267](https://doi.org/10.18653/v1/2023.acl-long.267). [Online]. Available: <https://aclanthology.org/2023.acl-long.267>.

- [50] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, “MetaICL: Learning to learn in context,” in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds., Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 2791–2809. DOI: [10.18653/v1/2022.naacl-main.201](https://doi.org/10.18653/v1/2022.naacl-main.201). [Online]. Available: <https://aclanthology.org/2022.naacl-main.201>.
- [51] Y. Wang, S. Mishra, P. Alipoormolabashi, *et al.*, “Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds., Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5085–5109. DOI: [10.18653/v1/2022.emnlp-main.340](https://doi.org/10.18653/v1/2022.emnlp-main.340). [Online]. Available: <https://aclanthology.org/2022.emnlp-main.340>.
- [52] Q. Dong, L. Li, D. Dai, *et al.*, “A survey on in-context learning,” *arXiv preprint arXiv:2301.00234*, 2022.
- [53] E. Tanwar, S. Dutta, M. Borthakur, and T. Chakraborty, “Multilingual LLMs are better cross-lingual in-context learners with alignment,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 6292–6307. DOI: [10.18653/v1/2023.acl-long.346](https://doi.org/10.18653/v1/2023.acl-long.346). [Online]. Available: <https://aclanthology.org/2023.acl-long.346>.
- [54] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, “What makes good in-context examples for GPT-3?” In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, E. Agirre, M. Apidianaki, and I. Vulić, Eds., Dublin, Ireland and Online: Association for Computational Linguistics, May 2022, pp. 100–114. DOI: [10.18653/v1/2022.deelio-1.10](https://doi.org/10.18653/v1/2022.deelio-1.10). [Online]. Available: <https://aclanthology.org/2022.deelio-1.10>.
- [55] T. Sorensen, J. Robinson, C. Rytting, *et al.*, “An information-theoretic approach to prompt engineering without ground truth labels,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 819–862. DOI: [10.18653/v1/2022.acl-long.60](https://doi.org/10.18653/v1/2022.acl-long.60). [Online]. Available: <https://aclanthology.org/2022.acl-long.60>.
- [56] H. Gonen, S. Iyer, T. Blevins, N. Smith, and L. Zettlemoyer, “Demystifying prompts in language models via perplexity estimation,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10136–10148. DOI: [10.18653/v1/2023.findings-emnlp.679](https://doi.org/10.18653/v1/2023.findings-emnlp.679). [Online]. Available: <https://aclanthology.org/2023.findings-emnlp.679>.
- [57] O. Rubin, J. Herzig, and J. Berant, “Learning to retrieve prompts for in-context learning,” in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds., Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 2655–2671.

- DOI: [10.18653/v1/2022.naacl-main.191](https://doi.org/10.18653/v1/2022.naacl-main.191). [Online]. Available: <https://aclanthology.org/2022.naacl-main.191>.
- [58] J. Ye, Z. Wu, J. Feng, T. Yu, and L. Kong, “Compositional exemplars for in-context learning,” in *International Conference on Machine Learning*, PMLR, 2023, pp. 39 818–39 833.
- [59] X. Wang, W. Zhu, and W. Y. Wang, “Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning,” *arXiv preprint arXiv:2301.11916*, p. 3, 2023.
- [60] Y. Zhang, S. Feng, and C. Tan, “Active example selection for in-context learning,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds., Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 9134–9148. DOI: [10.18653/v1/2022.emnlp-main.622](https://doi.org/10.18653/v1/2022.emnlp-main.622). [Online]. Available: <https://aclanthology.org/2022.emnlp-main.622>.
- [61] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing,” *ACM Comput. Surv.*, vol. 55, no. 9, 2023, ISSN: 0360-0300. DOI: [10.1145/3560815](https://doi.org/10.1145/3560815). [Online]. Available: <https://doi.org/10.1145/3560815>.
- [62] O. Honovich, U. Shaham, S. R. Bowman, and O. Levy, “Instruction induction: From few examples to natural language task descriptions,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1935–1952. DOI: [10.18653/v1/2023.acl-long.108](https://doi.org/10.18653/v1/2023.acl-long.108). [Online]. Available: <https://aclanthology.org/2023.acl-long.108>.
- [63] Y. Zhou, A. I. Muresanu, Z. Han, *et al.*, “Large language models are human-level prompt engineers,” *arXiv preprint arXiv:2211.01910*, 2022.
- [64] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, “Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8086–8098. DOI: [10.18653/v1/2022.acl-long.556](https://doi.org/10.18653/v1/2022.acl-long.556). [Online]. Available: <https://aclanthology.org/2022.acl-long.556>.
- [65] Y. Liu, J. Liu, X. Shi, Q. Cheng, and W. Lu, “Let’s learn step by step: Enhancing in-context learning ability with curriculum learning,” *arXiv preprint arXiv:2402.10738*, 2024.
- [66] Z. Li, Z. Xu, L. Han, *et al.*, “Implicit in-context learning,” *arXiv preprint arXiv:2405.14660*, 2024.
- [67] W. Zhou, Y. E. Jiang, R. Cotterell, and M. Sachan, “Efficient prompting via dynamic in-context learning,” *arXiv preprint arXiv:2305.11170*, 2023.
- [68] A. Chatterjee, E. Tanwar, S. Dutta, and T. Chakraborty, *Language models can exploit cross-task in-context learning for data-scarce novel tasks*, 2024. arXiv: [2405.10548](https://arxiv.org/abs/2405.10548) [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2405.10548>.
- [69] R. Agarwal, A. Singh, L. M. Zhang, *et al.*, *Many-shot in-context learning*, 2024. arXiv: [2404.11018](https://arxiv.org/abs/2404.11018) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2404.11018>.

- [70] A. Bertsch, M. Ivgi, U. Alon, J. Berant, M. R. Gormley, and G. Neubig, *In-context learning with long-context models: An in-depth exploration*, 2024. arXiv: 2405.00200 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2405.00200>.
- [71] S.-Y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," *arXiv preprint arXiv:2106.15772*, 2021.
- [72] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Synthetic prompting: Generating chain-of-thought demonstrations for large language models," in *International Conference on Machine Learning*, PMLR, 2023, pp. 30 706–30 775.
- [73] C. Anil, Y. Wu, A. Andreassen, *et al.*, "Exploring length generalization in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 38 546–38 556, 2022.
- [74] S. Diao, P. Wang, Y. Lin, and T. Zhang, "Active prompting with chain-of-thought for large language models," *arXiv preprint arXiv:2302.12246*, 2023.
- [75] B. Wang, S. Min, X. Deng, *et al.*, "Towards understanding chain-of-thought prompting: An empirical study of what matters," *arXiv preprint arXiv:2212.10001*, 2022.
- [76] X. Ye, S. Iyer, A. Celikyilmaz, V. Stoyanov, G. Durrett, and R. Pasunuru, "Complementary explanations for effective in-context learning," *arXiv preprint arXiv:2211.13892*, 2022.
- [77] F. Shi, M. Suzgun, M. Freitag, *et al.*, "Language models are multilingual chain-of-thought reasoners," *arXiv preprint arXiv:2210.03057*, 2022.
- [78] W. Chen, "Large language models are few(1)-shot table reasoners," in *Findings of the Association for Computational Linguistics: EACL 2023*, A. Vlachos and I. Augenstein, Eds., Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 1120–1130. DOI: 10.18653/v1/2023.findings-eacl.83. [Online]. Available: <https://aclanthology.org/2023.findings-eacl.83>.
- [79] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.
- [80] D. Zhou, N. Schärli, L. Hou, *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.
- [81] S. Sun, Y. Liu, S. Wang, C. Zhu, and M. Iyyer, "Pearl: Prompting large language models to plan and execute actions over long documents," *arXiv preprint arXiv:2305.14564*, 2023.
- [82] D. Dua, S. Gupta, S. Singh, and M. Gardner, "Successive prompting for decomposing complex questions," *arXiv preprint arXiv:2212.04092*, 2022.
- [83] S. Yao, D. Yu, J. Zhao, *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.
- [84] S. Hao, Y. Gu, H. Ma, *et al.*, "Reasoning with language model is planning with world model," *arXiv preprint arXiv:2305.14992*, 2023.
- [85] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," *arXiv preprint arXiv:2305.14078*, 2023.
- [86] X. Zhu, J. Wang, L. Zhang, *et al.*, "Solving math word problem via cooperative reasoning induced language models," *arXiv preprint arXiv:2210.16257*, 2022.

- [87] G. Dagan, F. Keller, and A. Lascarides, "Dynamic planning with a llm," *arXiv preprint arXiv:2308.06391*, 2023.
- [88] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents," *arXiv preprint arXiv:2302.01560*, 2023.
- [89] S. Yao, J. Zhao, D. Yu, *et al.*, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.
- [90] A. Madaan, N. Tandon, P. Gupta, *et al.*, "Self-refine: Iterative refinement with self-feedback," *arXiv preprint arXiv:2303.17651*, 2023.
- [91] S. Yao, D. Yu, J. Zhao, *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [92] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.
- [93] Y. Jiang, S. Bordia, Z. Zhong, C. Dognin, M. Singh, and M. Bansal, "Hover: A dataset for many-hop fact extraction and claim verification," *arXiv preprint arXiv:2011.03088*, 2020.
- [94] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "Fever: A large-scale dataset for fact extraction and verification," *arXiv preprint arXiv:1803.05355*, 2018.
- [95] Z. Yang, P. Qi, S. Zhang, *et al.*, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," *arXiv preprint arXiv:1809.09600*, 2018.
- [96] J. Bai, S. Bai, Y. Chu, *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.
- [97] K. Cobbe, V. Kosaraju, M. Bavarian, *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.
- [98] D. Hendrycks, C. Burns, S. Kadavath, *et al.*, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

Appendix A

Full Prompts

A.1 Baseline

Generate a pseudo program that describes the reasoning steps required to verify the claim step-by-step. You can call three functions in the program: 1. Question() to answer a question; 2. Verify() to verify a simple claim; 3. Predict() to predict the veracity label. Several examples are given as follows.

```
# The claim is that Howard University Hospital and Providence Hospital
are both located in Washington, D.C.
```

```
def program():
    fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
    fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
    label = Predict(fact_1 and fact_2)
```

```
# The claim is that An IndyCar race driver drove a Formula 1 car
designed by Peter McCool during the 2007 Formula One season.
```

```
def program():
    answer_1 = Question("Which Formula 1 car was designed by Peter
McCool during the 2007 Formula One season?")
    fact_1 = Verify(f"An IndyCar race driver drove the car {answer_1}.
")
    label = Predict(fact_1)
```

```
# The claim is that WWE Super Tuesday took place at an arena that
currently goes by the name TD Garden.
```

```
def program():
    answer_1 = Question("Which arena the WWE Super Tuesday took place?
")
    fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden.
")
    label = Predict(fact_1)
```

```
# The claim is that Talking Heads, an American rock band that was "one
of the most critically acclaimed bands of the 80's" is featured
in KSPN's AAA format.
```

```
def program():
```

```

fact_1 = Verify("Talking Heads is an American rock band that was '
one of the most critically acclaimed bands of the 80's'.")
fact_2 = Verify("Talking Heads is featured in KSPN's AAA format.")
label = Predict(fact_1 and fact_2)

```

The claim is that The model of car Trevor Bayne drives was introduced for model year 2006. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup.

```

def program():
    answer_1 = Question("Which model of car is driven by Trevor Bayne?")
    fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
    answer_2 = Question("Who is the Rookie of The Year in the 1997 CART season?")
    fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR Sprint Cup.")
    label = predict(fact_1 and fact_2)

```

The claim is that The song recorded by Fergie that was produced by Polow da Don and was followed by Life Goes On was M.I.L.F.\$.

```

def program():
    fact_1 = Verify("M.I.L.F.$ was recorded by Fergie.")
    fact_2 = Verify("M.I.L.F.$ was produced by Polow da Don.")
    fact_3 = Verify("M.I.L.F.$ was followed by Life Goes On.")
    label = Predict(fact_1 and fact_2 and fact_3)

```

The claim is that Gina Bramhill was born in a village. The 2011 population of the area that includes this village was 167,446.

```

def program():
    answer_1 = Question("Which village was Gina Bramhill born in?")
    fact_1 = Verify(f"The 2011 population of the area that includes {answer_1} was 167,446.")
    label = Predict(fact_1)

```

The claim is that Don Ashley Turlington graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish.

```

def program():
    fact_1 = Verify("Saint Joseph's College is a private Catholic liberal arts college in Standish.")
    fact_2 = Verify(f"Don Ashley Turlington graduated from Saint Joseph's College.")
    label = Predict(fact_1 and fact_2)

```

The claim is that In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach.

```

def program():
    answer_1 = Question("Which 2004 Hockey film was produced by a former major league baseball pitcher?")
    fact_1 = Verify("Kurt Russell played the USA coach in the film {answer_1}.")
    label = Predict(fact_1)

```

```
# The claim is that Jack McFarland is the best known role of the host
of the 64th Annual Tony Awards.
def program():
    answer_1 = Question("Who are the hosts of the 64th Annual Tony
Awards?")
    fact_1 = Verify(f"Jack McFarland is the best known role of {
answer_1}.")
    label = Predict(fact_1)
```

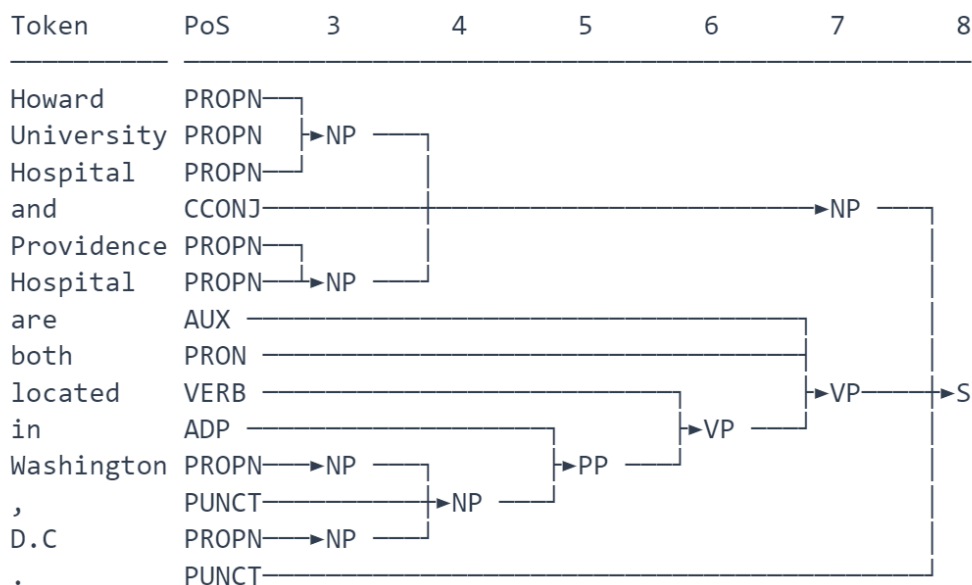
```
# The claim is that [[CLAIM]]
def program():
```

A.2 No Reasoning with No Supervision(NR-NS)

Given a claim and its constituency tree, generate analysis about how to decompose the claim, and a pseudo program that describes the reasoning steps required to verify the claim step-by-step. You can call three functions in the program: 1. Question() to answer a question; 2. Verify() to verify a simple claim; 3. Predict() to predict the veracity label. Several examples are given as follows.

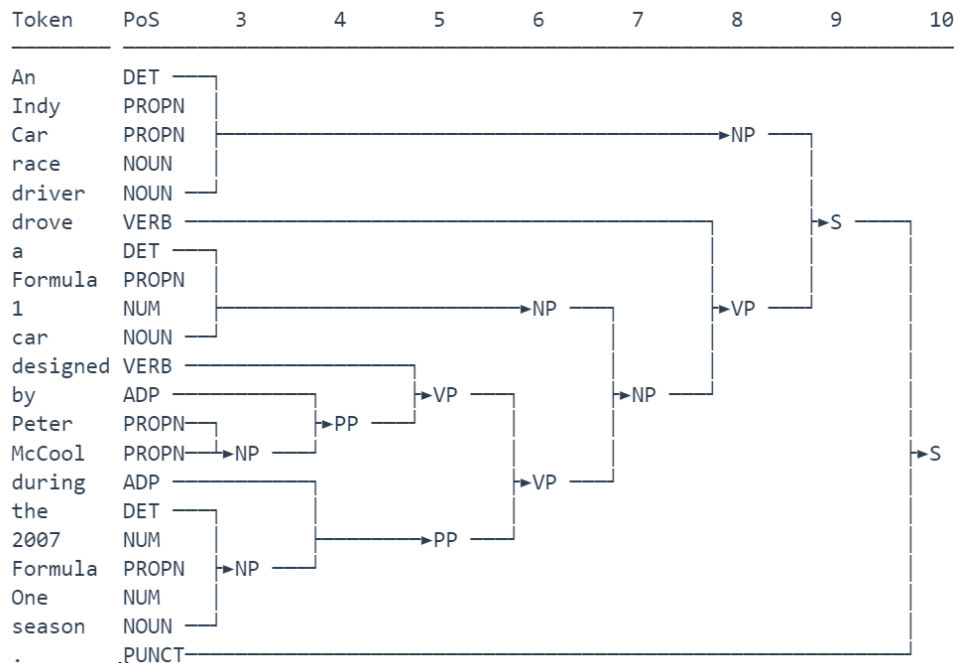
```
# The claim is that Howard University Hospital and Providence Hospital
are both located in Washington, D.C.
```

Constituency parsing tree:



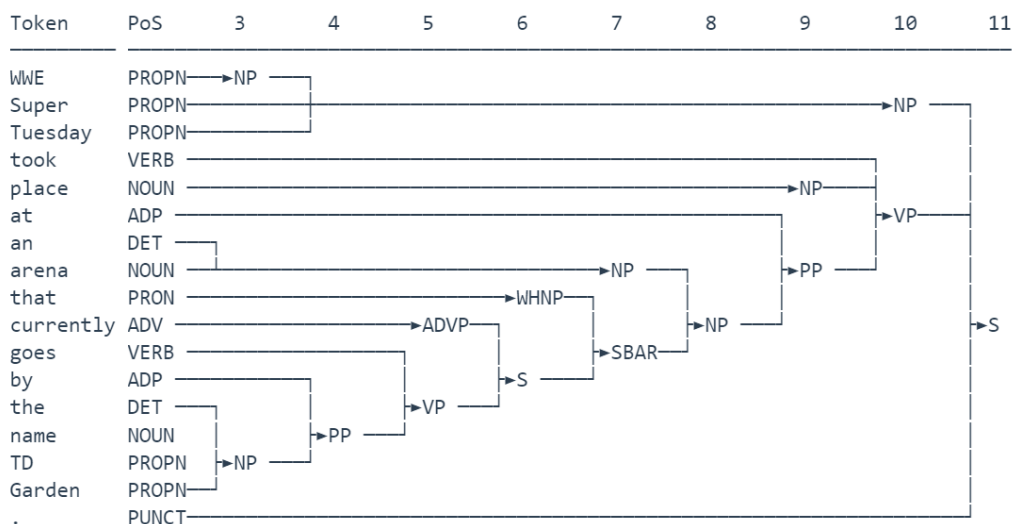
```
def program():
    fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
    fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
    label = Predict(fact_1 and fact_2)
```

The claim is that An IndyCar race driver drove a Formula 1 car designed by Peter McCool during the 2007 Formula One season.
Constituency parsing tree:



```
def program():
    answer_1 = Question("Which Formula 1 car was designed by Peter
    McCool during the 2007 Formula One season?")
    fact_1 = Verify(f"An IndyCar race driver drove the car {answer_1
    }.")
    label = Predict(fact_1)
```

The claim is that WWE Super Tuesday took place at an arena that currently goes by the name TD Garden.
Constituency parsing tree:



```
def program():
```

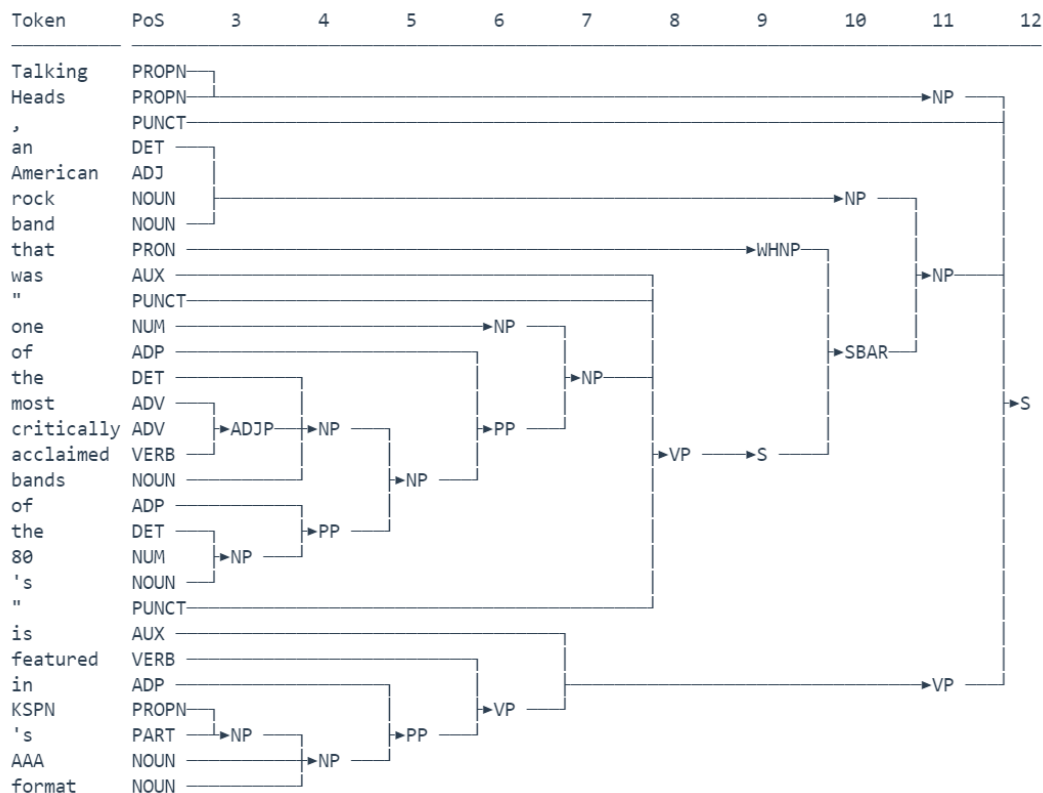
```

answer_1 = Question("Which arena the WWE Super Tuesday took place
?")
fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden
.")
label = Predict(fact_1)

```

The claim is that Talking Heads, an American rock band that was "one of the most critically acclaimed bands of the 80's" is featured in KSPN's AAA format.

Constituency parsing tree:



```

def program():
    fact_1 = Verify("Talking Heads is an American rock band that was '
one of the most critically acclaimed bands of the 80's'.")
    fact_2 = Verify("Talking Heads is featured in KSPN's AAA format.")
    label = Predict(fact_1 and fact_2)

```

The claim is that The model of car Trevor Bayne drives was introduced for model year 2006. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup.

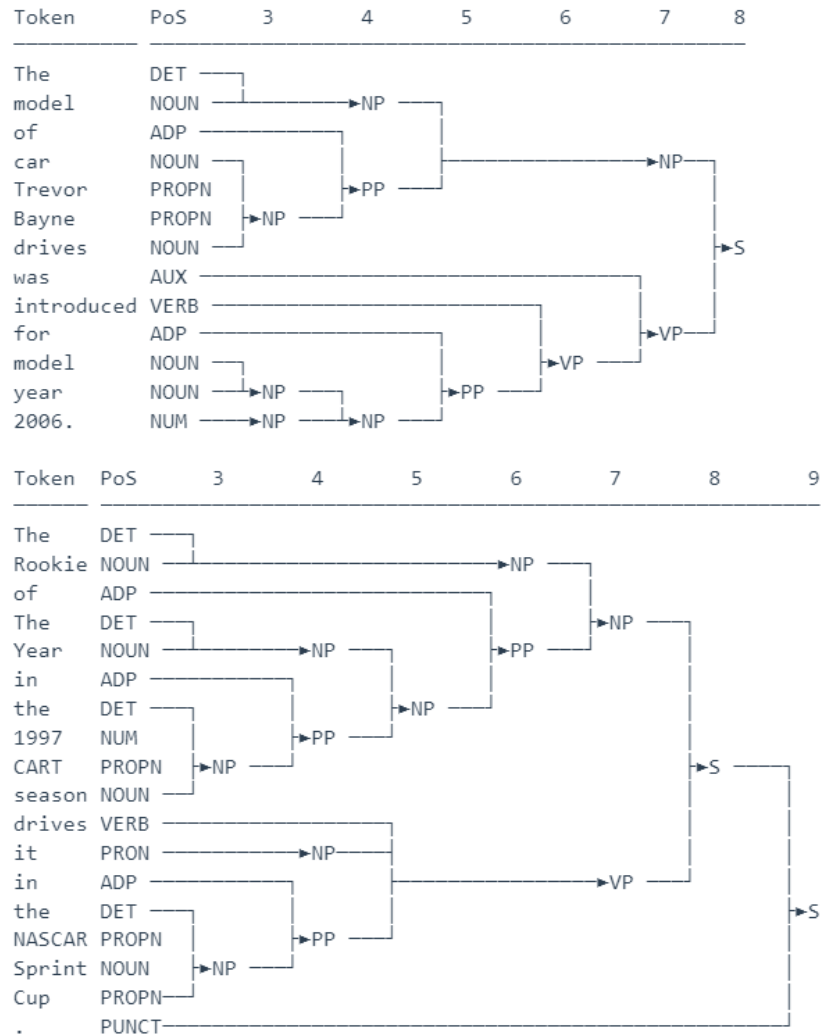
Constituency parsing tree:

ewpage

```

def program():
    answer_1 = Question("Which model of car is driven by Trevor Bayne
?")
    fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
    answer_2 = Question("Who is the Rookie of The Year in the 1997
CART season?")

```



```
fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR
Sprint Cup.")
label = predict(fact_1 and fact_2)
```

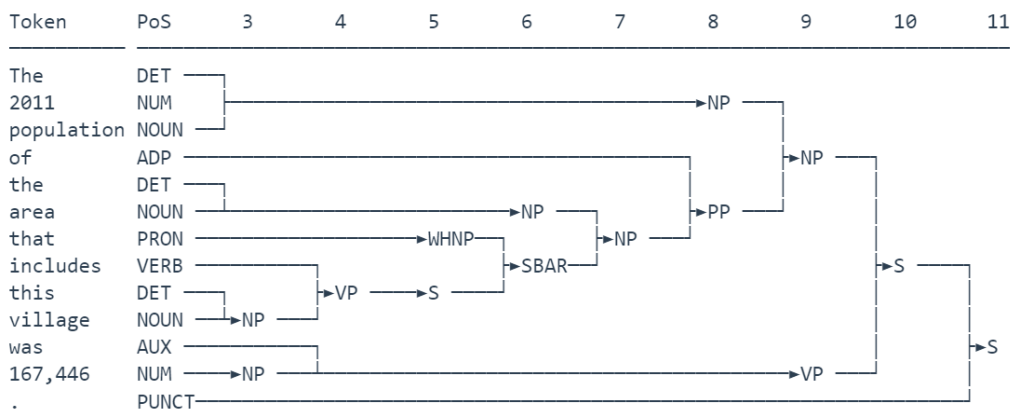
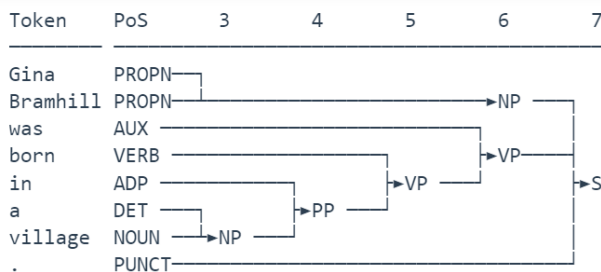
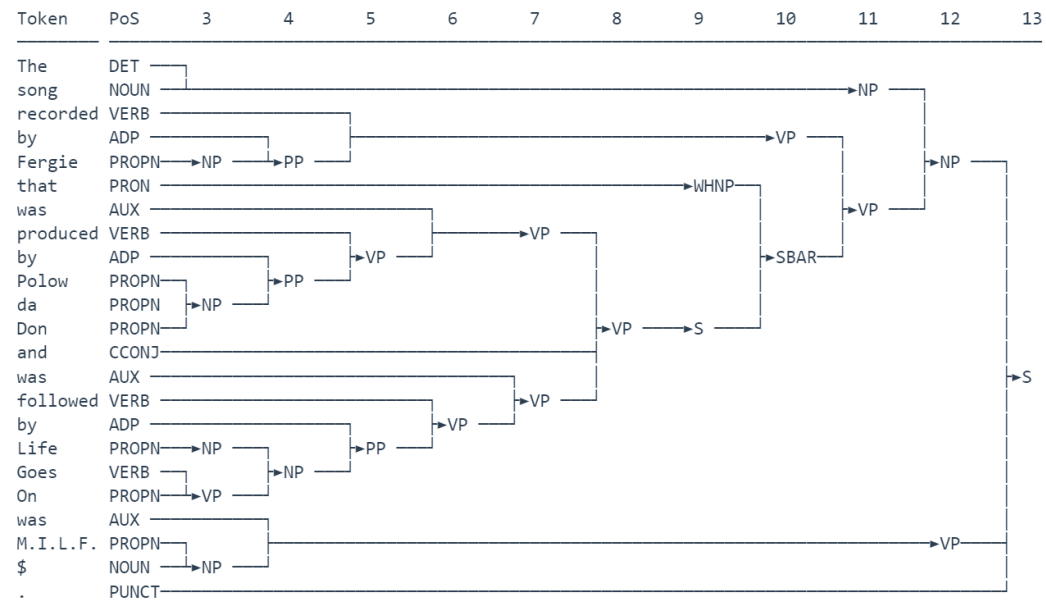
```
# The claim is that The song recorded by Fergie that was produced by
Polow da Don and was followed by Life Goes On was M.I.L.F.$.
Constituency parsing tree:
```

```
ewpage
```

```
def program():
    fact_1 = Verify("M.I.L.F.$ was recorded by Fergie.")
    fact_2 = Verify("M.I.L.F.$ was produced by Polow da Don.")
    fact_3 = Verify("M.I.L.F.$ was was followed by Life Goes On.")
    label = Predict(fact_1 and fact_2 and fact_3)
```

```
# The claim is that Gina Bramhill was born in a village. The 2011
population of the area that includes this village was 167,446.
Constituency parsing tree:
```

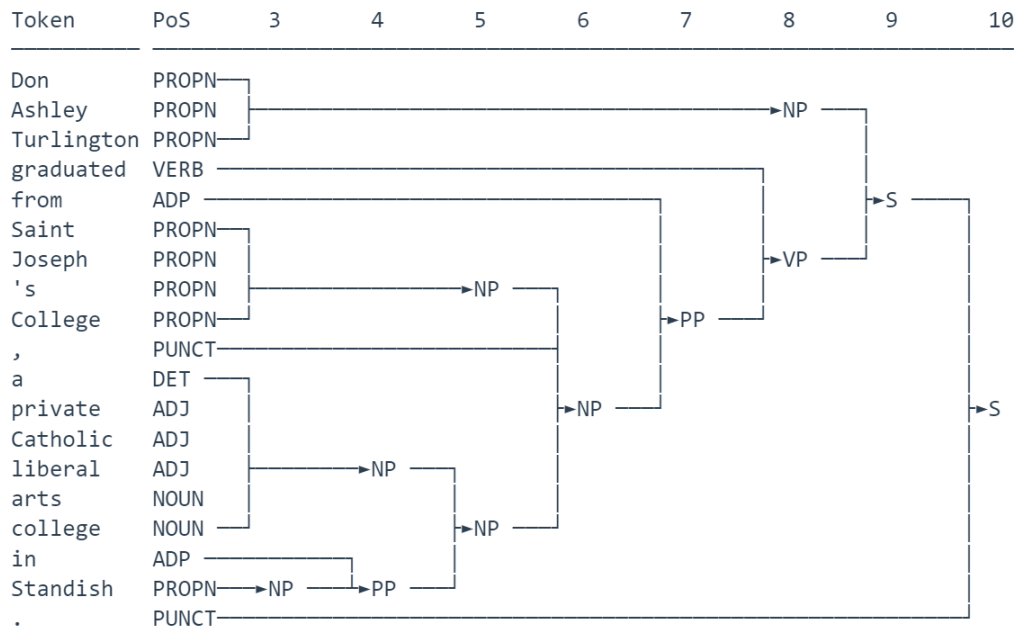
```
def program():
    answer_1 = Question("Which village was Gina Bramhill born in?")
```



```
fact_1 = Verify(f"The 2011 population of the area that includes {
answer_1} was 167,446.")
label = Predict(fact_1)
```

The claim is that Don Ashley Turlington graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish. Constituency parsing tree:

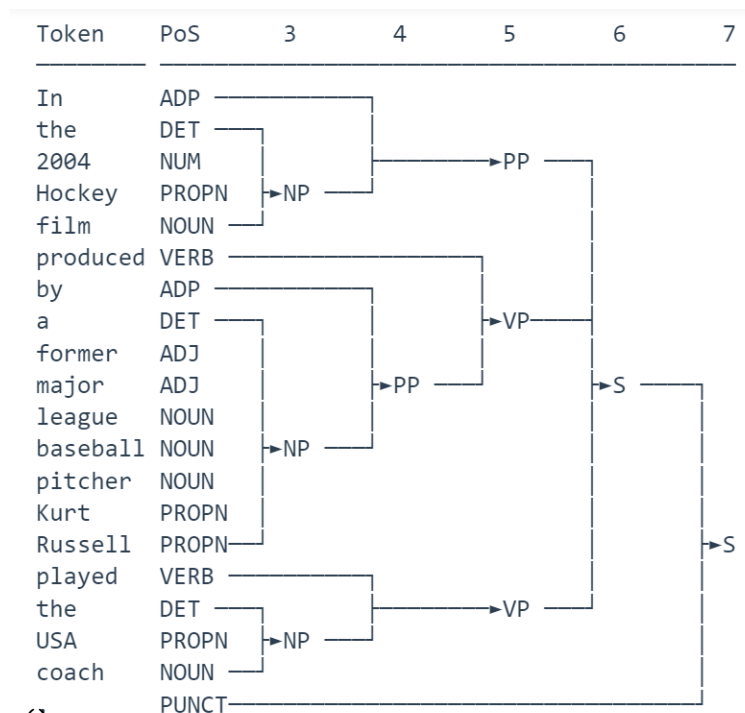
```
def program():
    fact_1 = Verify("Saint Joseph's College is a private Catholic
liberal arts college in Standish.")
    fact_2 = Verify(f"Don Ashley Turlington graduated from Saint
Joseph's College.")
    label = Predict(fact_1 and fact_2)
```



The claim is that In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach.

Constituency parsing tree:

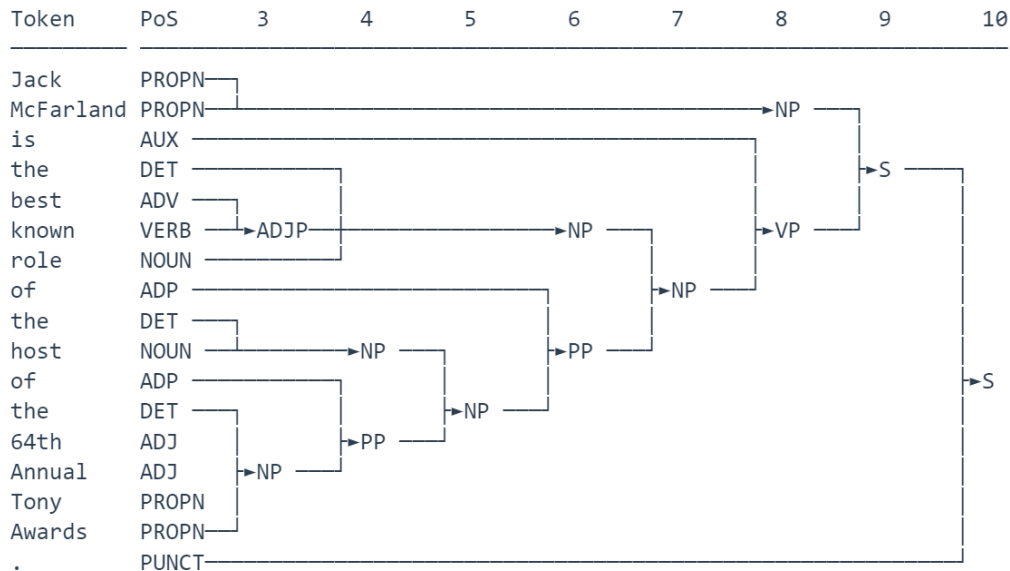
ewpage



```
def program():
    answer_1 = Question("Which 2004 Hockey film was produced a former
    major league baseball pitcher?")
    fact_1 = Verify("Kurt Russell played the USA coach in the film {
    answer_1}.")
    label = Predict(fact_1)
```


The claim is that Jack McFarland is the best known role of the host of the 64th Annual Tony Awards.

Constituency parsing tree:



```
def program():
    answer_1 = Question("Who are the hosts of the 64th Annual Tony
Awards?")
    fact_1 = Verify(f"Jack McFarland is the best known role of {
answer_1}.")
    label = Predict(fact_1)
```

The claim is that [[CLAIM]]

Constituency parsing tree:

[[TREE]]

```
def program():
```

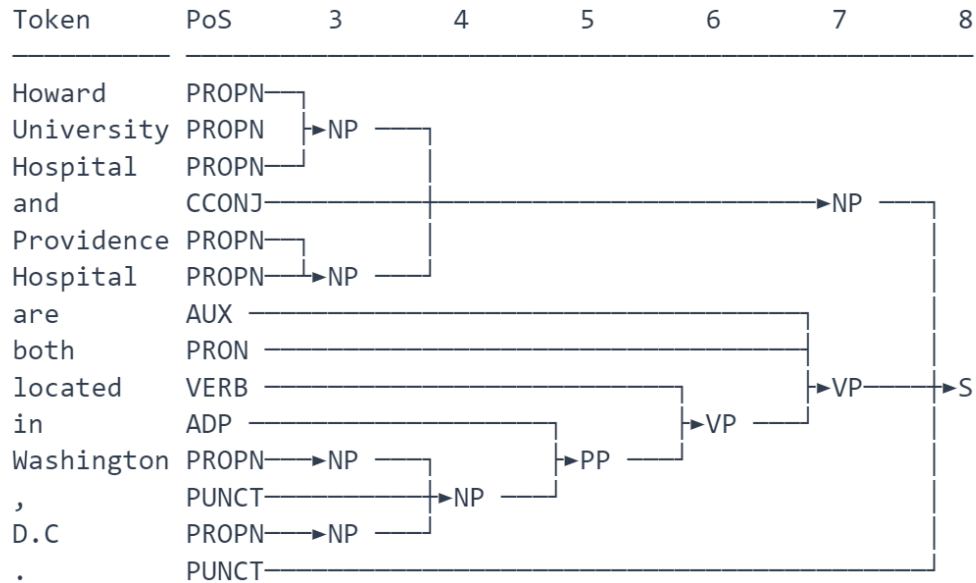
A.3 No Reasoning with Supervision (NR-S)

```
[
{
  "role": "system",
  "content": "You have strong ability of reasoning. You only need to
generate an executable pseudo program within simple structure
that reflect the reasoning steps required to verify the claim step
-by-step. The constituency tree of the claim is provided to help
you understand the claim structure and decompose it. You can only
call three functions in the program: 1. Question() to answer a
question; 2. Verify() to verify a simple claim; 3. Predict() to
predict the veracity label, the only logical operators can be used
is 'and'."
},
{
```

```

"role": "user",
"content": "The claim is that Howard University Hospital and
Providence Hospital are both located in Washington, D.C.. The
constituency parsing tree is:

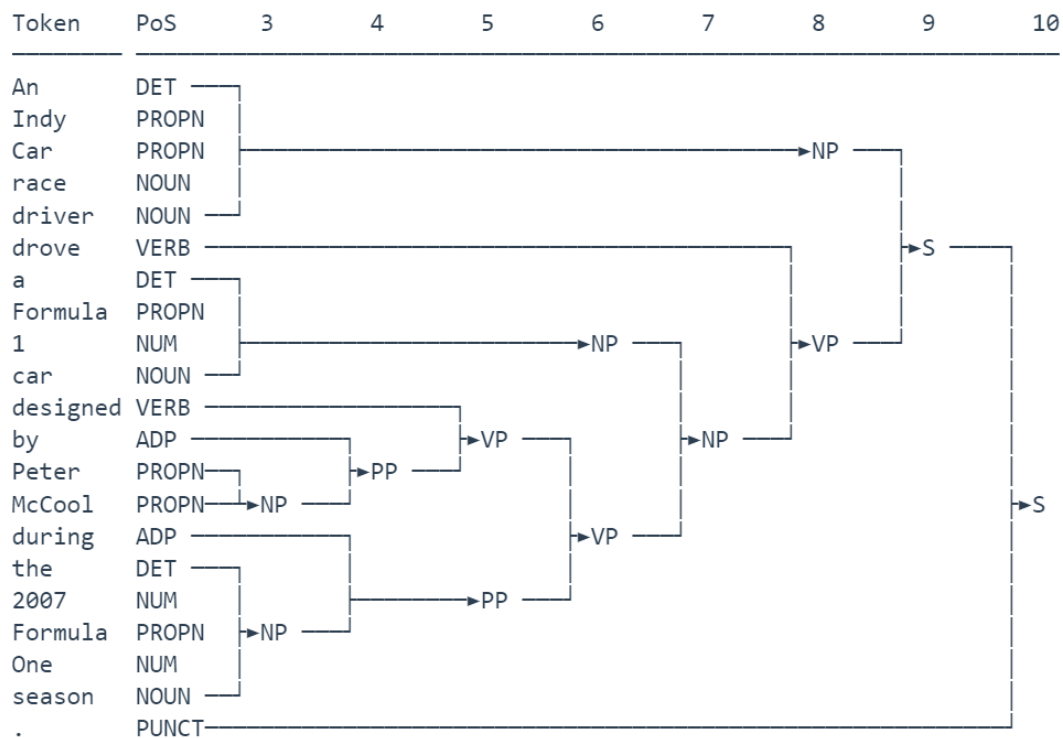
```



```

"
},
{
  "role": "assistant",
  "content": "`pseudo # Decompose the claim into simpler sub-
claims sub_claim1 = "Howard University Hospital is located in
Washington, D.C." sub_claim2 = "Providence Hospital is located in
Washington, D.C.""
},
{
  "role": "user",
  "content": "The correct answer is: def program():      fact_1 =
Verify("Howard University Hospital is located in Washington, D.C
.")      fact_2 = Verify("Providence Hospital is located in
Washington, D.C.")      label = Predict(fact_1 and fact_2).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
  "role": "user",
  "content": "The claim is that An IndyCar race driver drove a
Formula 1 car designed by Peter McCool during the 2007 Formula One
season. The constituency parsing tree is:
"
},
{
  "role": "assistant",

```

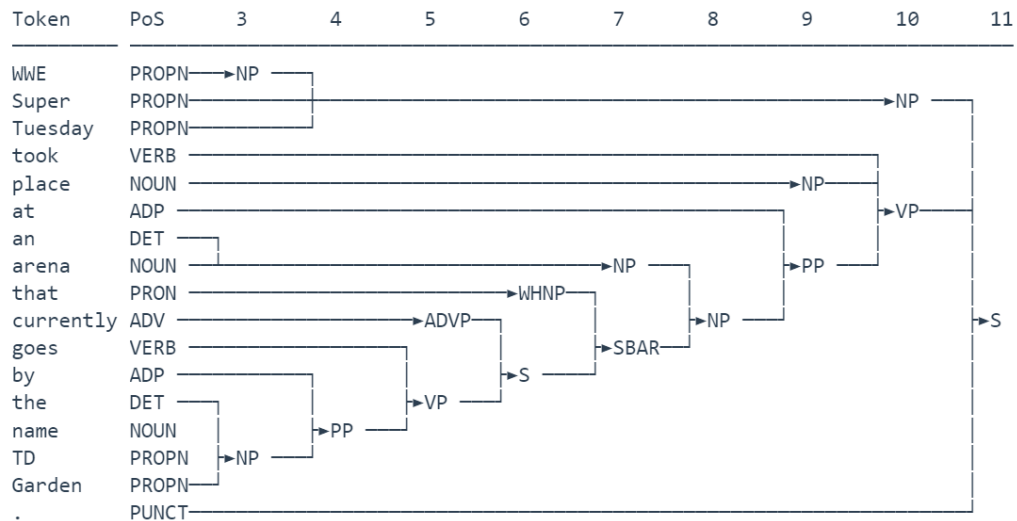


```

"content": "```pseudodef program():      # Decompose the claim
into simpler sub-claims      sub_claim1 = "An IndyCar race driver
drove a Formula 1 car during the 2007 Formula One season."
sub_claim2 = "The Formula 1 car was designed by Peter McCool.""
},
{
  "role": "user",
  "content": "The correct answer is: def program():      answer_1 =
Question("Which Formula 1 car was designed by Peter McCool during
the 2007 Formula One season?")      fact_1 = Verify(f"An IndyCar
race driver drove the car {answer_1}.")      label = Predict(
fact_1). Carefully examine the differences between your answers
and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},
{
  "role": "user",
  "content": "The claim is that  WWE Super Tuesday took place at an
arena that currently goes by the name TD Garden. The constituency
parsing tree is:

"
},
{
  "role": "assistant",

```

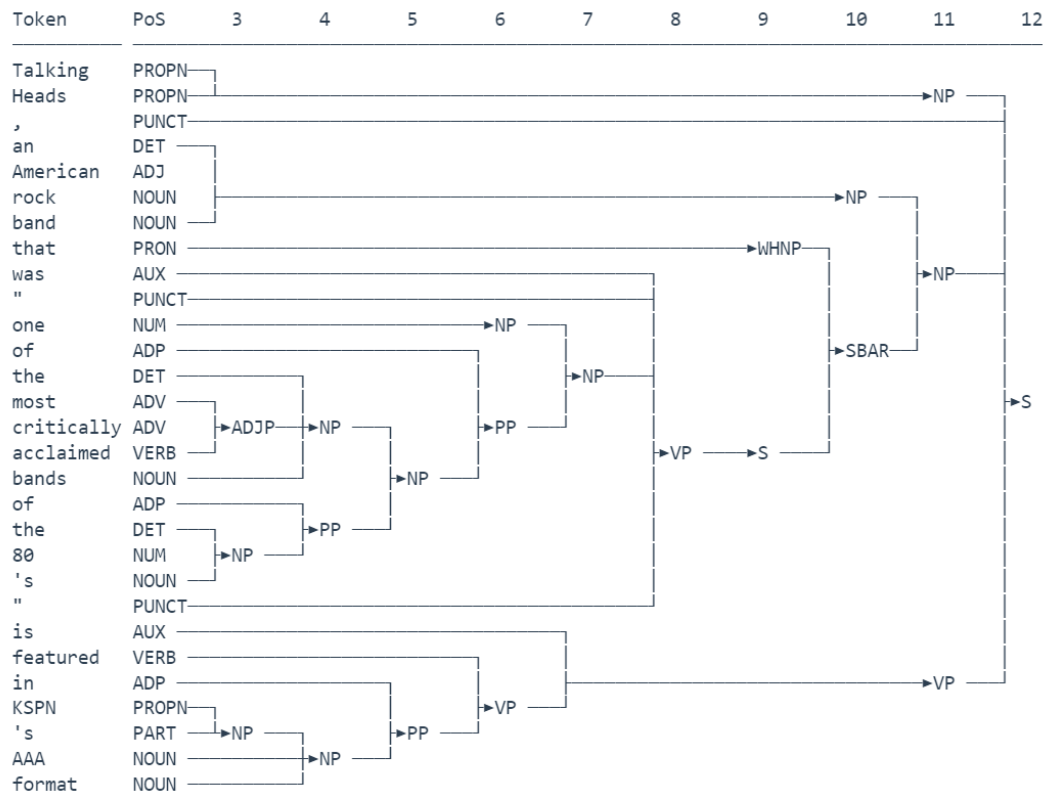


```

"content": "```pseudo def program():      answer_1 = Question("
What is the current name of the arena where WWE Super Tuesday took
place?")      fact_1 = Verify(f"The arena is currently called {
answer_1}.")      fact_2 = Verify("WWE Super Tuesday took place at
the arena.")      label = Predict(fact_1 and fact_2)  ```"
},
{
  "role": "user",
  "content": "The correct answer is: def program():      answer_1 =
Question("Which arena the WWE Super Tuesday took place?")
fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden
.")      label = Predict(fact_1). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
  "role": "user",
  "content": "The claim is that Talking Heads, an American rock
band that was "one of the most critically acclaimed bands of the
80's" is featured in KSPN's AAA format. The constituency parsing
tree is:

"
},
{
  "role": "assistant",
  "content": "```pseudo def program():      answer_1 = Question("
Which band is featured in KSPN's AAA format?")      fact_1 =
Verify(f"The band is Talking Heads.")      fact_2 = Verify("
Talking Heads is an American rock band that was one of the most
critically acclaimed bands of the 80's.")      label = Predict(
fact_1 and fact_2)  ```"
},
{
  "role": "user",

```

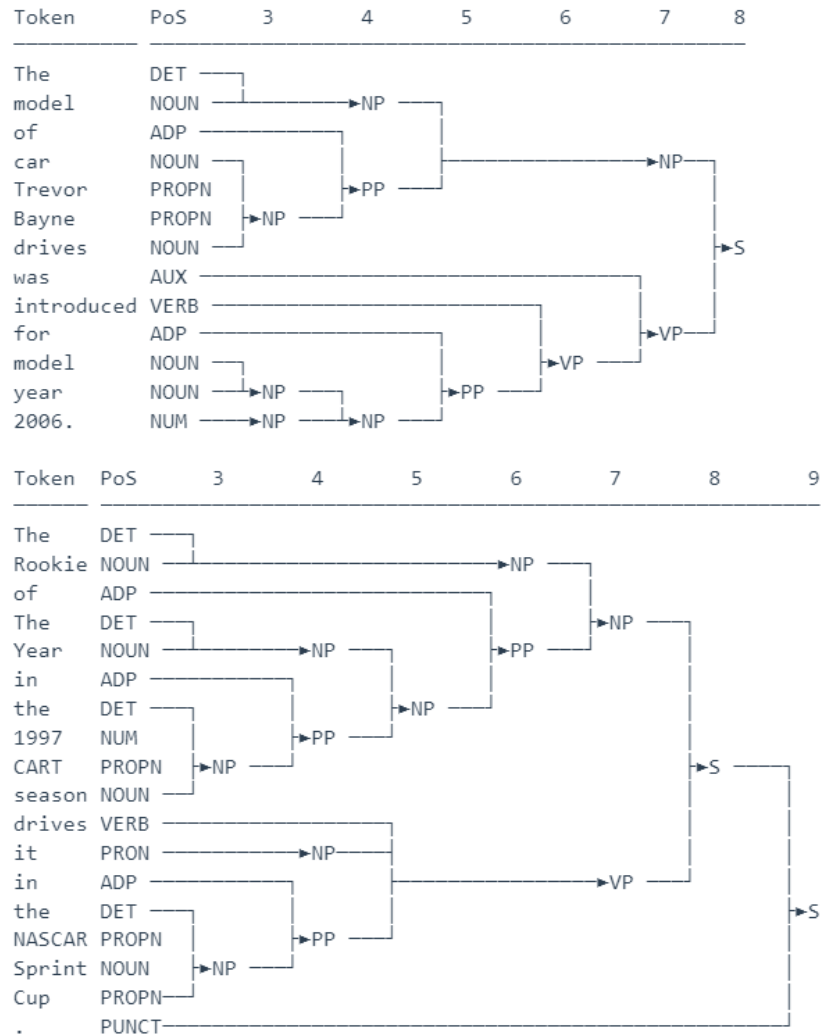


```

"content": "The correct answer is: def program():      fact_1 =
Verify("Talking Heads is an American rock band that was 'one of
the most critically acclaimed bands of the 80's'.")      fact_2 =
Verify("Talking Heads is featured in KSPN's AAA format.")
label = Predict(fact_1 and fact_2). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
"role": "user",
"content": "The claim is that The model of car Trevor Bayne drives
was introduced for model year 2006. The Rookie of The Year in the
1997 CART season drives it in the NASCAR Sprint Cup. The
constituency parsing tree is:

"
},
{
"role": "assistant",
"content": "`pseudo def program():      answer_1 = Question("
What model of car does Trevor Bayne drive?")      fact_1 = Verify("
f"The model of car Trevor Bayne drives was introduced for model
year 2006.")      fact_2 = Verify(f"The Rookie of The Year in the
1997 CART season drives the car in the NASCAR Sprint Cup.")
label = Predict(fact_1 and fact_2)  "`"
},
{
"role": "user",

```

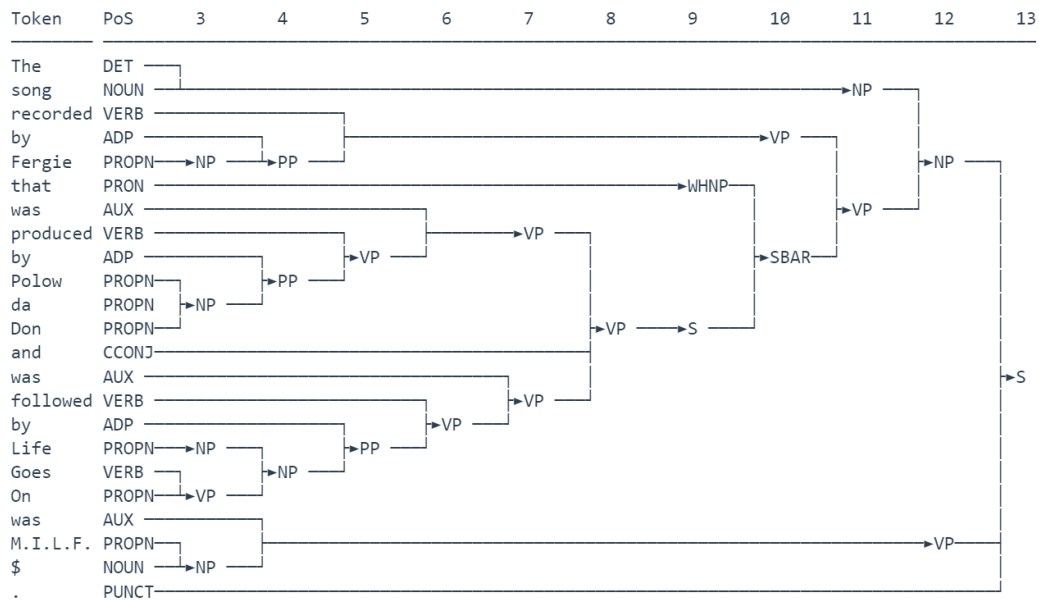


```

"content": "The correct answer is: def program():      answer_1 =
Question("Which model of car is driven by Trevor Bayne?")
fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
    answer_2 = Question("Who is the Rookie of The Year in the
1997 CART season?")      fact_2 = Verify(f"{answer_2} drives {
answer_1} in the NASCAR Sprint Cup.")      label = predict(fact_1
and fact_2). Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},
{
    "role": "user",
    "content": "The claim is that The song recorded by Fergie that
was produced by Polow da Don and was followed by Life Goes On was
M.I.L.F.$. The constituency parsing tree is:

"
},
{
    "role": "assistant",

```

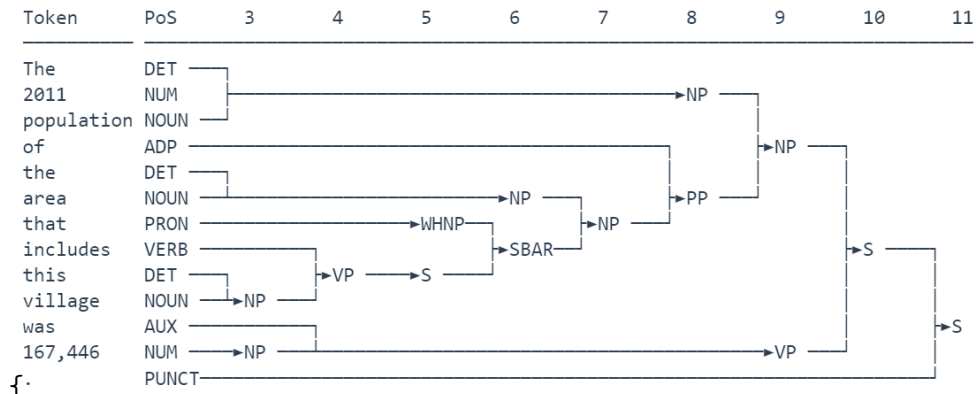
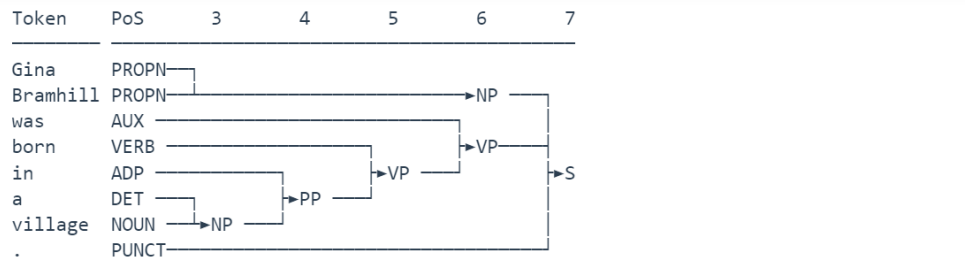


```

"content": "`pseudo def program():      answer_1 = Question("
Which song was recorded by Fergie and produced by Polow da Don?")
      fact_1 = Verify(f"The song recorded by Fergie that was
produced by Polow da Don was {answer_1}.")      fact_2 = Verify(f"
The song followed by Life Goes On was {answer_1}.")      label =
Predict(fact_1 and fact_2)  "`"
},
{
  "role": "user",
  "content": "The correct answer is: def program():      fact_1 =
Verify("M.I.L.F.$ was recorded by Fergie.")      fact_2 = Verify("
M.I.L.F.$ was produced by Polow da Don.")      fact_3 = Verify("M.
I.L.F.$ was was followed by Life Goes On.")      label = Predict(
fact_1 and fact_2 and fact_3)      . Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
  "role": "user",
  "content": "The claim is that Gina Bramhill was born in a village.
The 2011 population of the area that includes this village was
167,446. The constituency parsing tree is:

"
},
{
  "role": "assistant",
  "content": "`pseudo def program():      fact_1 = Verify("Gina
Bramhill was born in a village.")      fact_2 = Verify("The 2011
population of the area that includes this village was 167,446.")
      label = Predict(fact_1 and fact_2)  "`"
},

```



```

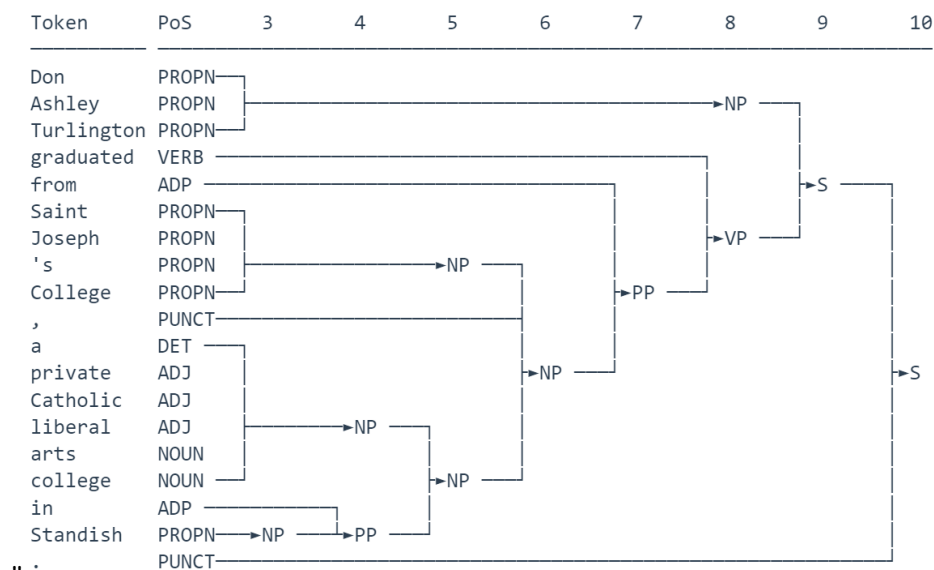
{
  "role": "user",
  "content": "The correct answer is: def program():      answer_1 =
Question("Which village was Gina Bramhill born in?")      fact_1 =
Verify(f"The 2011 population of the area that includes {answer_1}
was 167,446.")      label = Predict(fact_1). Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{

```

```

  "role": "user",
  "content": "The claim is that Don Ashley Turlington graduated
from Saint Joseph's College, a private Catholic liberal arts
college in Standish. The constituency parsing tree is:

```




```

},
{
  "role": "assistant",
  "content": "```pseudo def program():      fact_1 = Verify("Don
Ashley Turlington graduated from Saint Joseph's College.")
fact_2 = Verify("Saint Joseph's College is a private Catholic
liberal arts college in Standish.")      label = Predict(fact_1
and fact_2)  ```"
},
{
  "role": "user",
  "content": "The correct answer is: def program():      fact_1 =
Verify("Saint Joseph's College is a private Catholic liberal arts
college in Standish.")      fact_2 = Verify(f"Don Ashley
Turlington graduated from Saint Joseph's College.")      label =
Predict(fact_1 and fact_2). Carefully examine the differences
between your answers and the standard ones. Learn from the correct
decomposing approaches and formats, and apply these insights to
your next claim."
},
{
  "role": "user",
  "content": "The claim is that In the 2004 Hockey film produced by
a former major league baseball pitcher Kurt Russell played the USA
coach. The constituency parsing tree is:

```

Token	PoS	3	4	5	6	7
In	ADP					
the	DET					
2004	NUM					
Hockey	PROPN					
film	NOUN					
produced	VERB					
by	ADP					
a	DET					
former	ADJ					
major	ADJ					
league	NOUN					
baseball	NOUN					
pitcher	NOUN					
Kurt	PROPN					
Russell	PROPN					
played	VERB					
the	DET					
USA	PROPN					
coach	NOUN					
.	PUNCT					

```

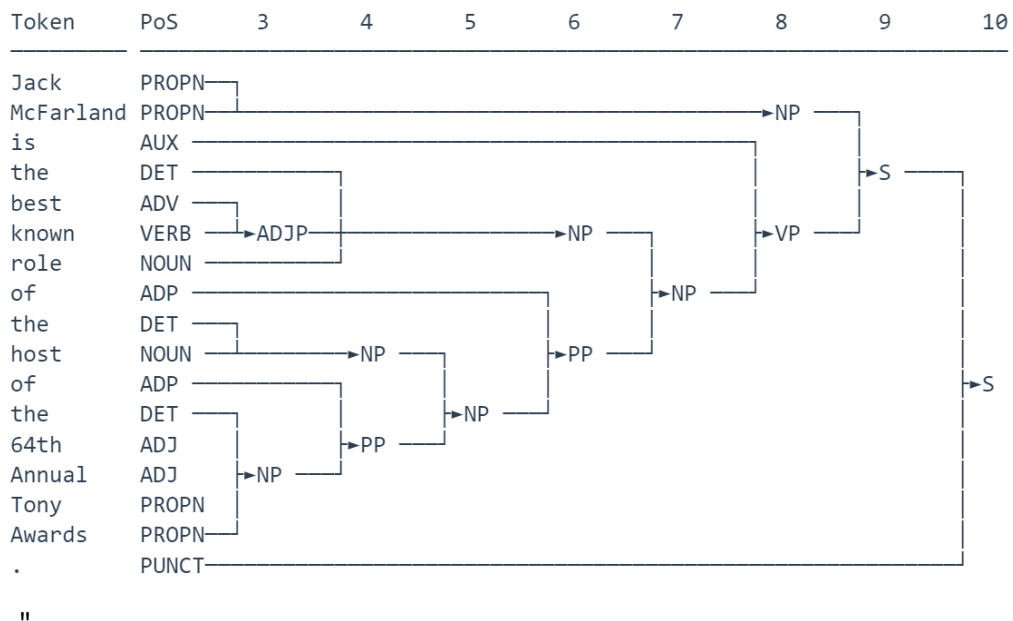
"
},
{
  "role": "assistant",

```

```

"content": "```pseudo def program():      answer_1 = Question("
Which 2004 Hockey film was produced by a former major league
baseball pitcher?")      fact_1 = Verify(f"Kurt Russell played the
USA coach in the film {answer_1}.")      label = Predict(fact_1)
```"
},
{
 "role": "user",
 "content": "The correct answer is: def program(): answer_1 =
Question("Which 2004 Hockey film was produced a former major
league baseball pitcher?") fact_1 = Verify("Kurt Russell
played the USA coach in the film {answer_1}.") label =
Predict(fact_1). Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},
{
 "role": "user",
 "content": "The claim is that Jack McFarland is the best known
role of the host of the 64th Annual Tony Awards. The constituency
parsing tree is:

```



```

"
},
{
 "role": "assistant",
 "content": "```pseudo def program(): answer_1 = Question("
Who is the host of the 64th Annual Tony Awards?") fact_1 =
Verify(f"Jack McFarland is the best known role of {answer_1}.")
 label = Predict(fact_1) ```"
},
{
 "role": "user",

```

```

"content": "The correct answer is: def program(): answer_1 =
Question("Who are the hosts of the 64th Annual Tony Awards?")
 fact_1 = Verify(f"Jack McFarland is the best known role of {
answer_1}.") label = Predict(fact_1). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
}
]

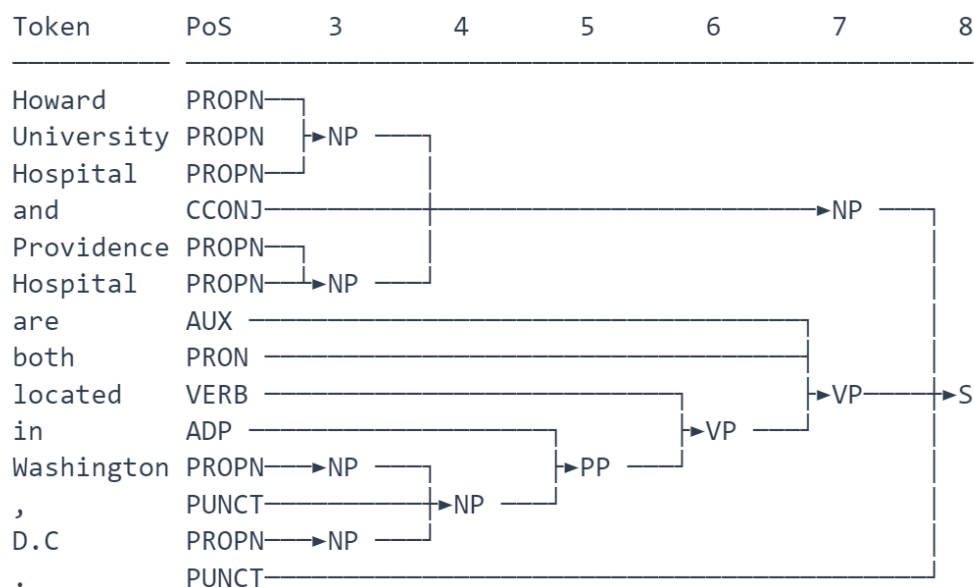
```

## A.4 Chain-of-Thought with No Supervision(CoT-NS)

Given a claim and its constituency tree, generate analysis about how to decompose the claim, and a pseudo program that describes the reasoning steps required to verify the claim step-by-step. You can call three functions in the program: 1. Question() to answer a question; 2. Verify() to verify a simple claim; 3. Predict() to predict the veracity label. Several examples are given as follows.

# The claim is that Howard University Hospital and Providence Hospital are both located in Washington, D.C.

Constituency parsing tree:



Analysis: At the top, the sentence is divided into a large noun phrase (NP) and a verb phrase (VP). The noun phrase itself is a compound structure, made up of two separate entities: "Howard University Hospital" and "Providence Hospital." These entities are connected by the conjunction "and," indicating that the action described in the verb phrase applies equally to both entities.

Moving to the verb phrase, it consists of the auxiliary verb "are," the adverb "both," the main verb "located," and a prepositional phrase "in Washington, D.C." This structure tells us that the state of being located is shared by both subjects of the compound noun phrase. The prepositional phrase at the end of the sentence specifies the location, indicating that both hospitals are situated in Washington, D.C.

In breaking down this sentence, we observe how each part of the structure contributes to the overall meaning. The conjunction in the noun phrase effectively binds the two hospitals into a shared circumstance, while the components of the verb phrase distribute this circumstance across both subjects equally. Thus, using the constituency tree helps to visually and logically segment the sentence into digestible parts without altering the intended meaning. This detailed analysis shows how both hospitals, though mentioned separately, are unified in their description, emphasizing their collective presence in the same geographical location.

From this analysis, we can derive two simpler sub-sentences: 1. Howard University Hospital is located in Washington, D.C. 2. Providence Hospital is also located in Washington, D.C.

Based on this information, we can transform each fact into a Verify() function. The first Verify() function checks if the "Howard University Hospital is located in Washington, D.C." and the second Verify() function checks if "Providence Hospital is located in Washington, D.C.". Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two Verify() functions. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

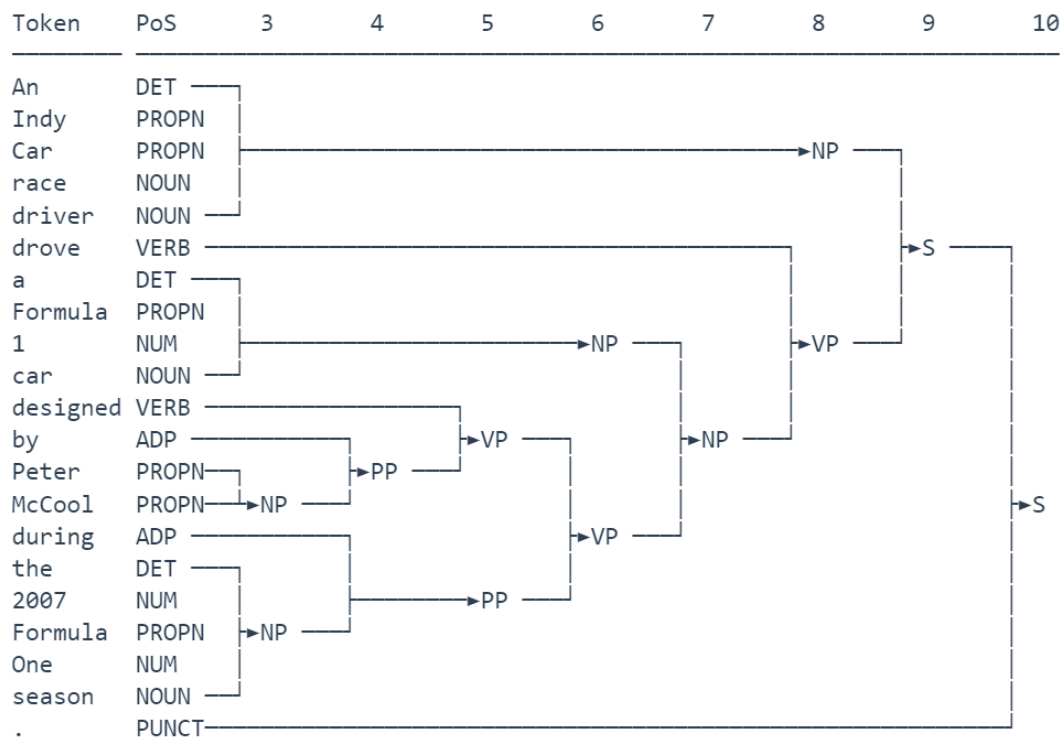
Program code:

```
fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
label = Predict(fact_1 and fact_2)
```

# The claim is that An IndyCar race driver drove a Formula 1 car designed by Peter McCool during the 2007 Formula One season.  
Constituency parsing tree:

ewpage

Analysis: We use a top-down approach to analyze this sentence. We start by identifying the highest node in the parsing tree, which categorizes the entire sentence into two primary components: a noun phrase (NP) and a verb phrase (VP).



**Noun Phrase (NP):** At the top level, the NP "An IndyCar race driver" is selected. This phrase is identified as the subject of the sentence. The NP combines a determiner "An" with a compound noun "IndyCar race driver". This NP sets the stage by identifying who is performing the action, grounding the sentence in the world of IndyCar racing.

**Verb Phrase (VP):** Following the subject, we examine the VP "drove a Formula 1 car designed by Peter McCool during the 2007 Formula One season." This complex VP begins with the verb "drove," which is the main action connecting the subject to the direct object "a Formula 1 car." This object itself extends into a more detailed description through additional nested structures: 1. The phrase "designed by Peter McCool" is an embedded clause that modifies "a Formula 1 car," specifying that the car was designed by Peter McCool. Here, "designed" acts as a verb in a smaller verb phrase, which includes a prepositional phrase "by Peter McCool." "Peter McCool" serves as a noun phrase within this prepositional phrase, attributing the design to him. 2. The final part of the VP, "during the 2007 Formula One season," provides the temporal context. This is structured as a prepositional phrase containing "the 2007 Formula One season," which acts as a noun phrase detailing when the action took place.

By decomposing the original sentence, we can derive two simplified sub-sentences that preserve the meaning but clarify the structure: 1. An IndyCar race driver drove a Formula 1 car. 2. The car was designed by Peter McCool during the 2007 Formula One season.

We can pose the second part of the claim as a question using the Question() function: "Which Formula 1 car was designed by Peter McCool during the 2007 Formula One season?". This question aims to identify the specific car that the IndyCar race driver drove. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "An IndyCar race driver drove the car {answer\_1}". This is done using the Verify() function and stored in the variable fact\_1. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (fact\_1) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

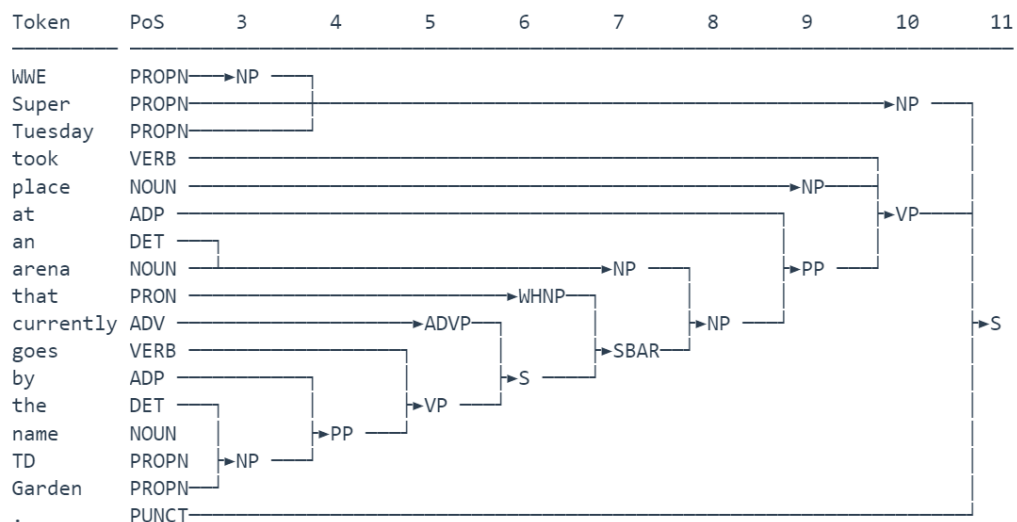
Program code:

```
answer_1 = Question("Which Formula 1 car was designed by Peter
McCool during the 2007 Formula One season?")
fact_1 = Verify(f"An IndyCar race driver drove the car {answer_1
}.".)
label = Predict(fact_1)
```

# The claim is that WWE Super Tuesday took place at an arena that currently goes by the name TD Garden.

Constituency parsing tree:

ewpage



**Analysis:** We use a top-down approach to analyze this sentence. The sentence's structure is initially split into a noun phrase (NP) "WWE Super Tuesday" and a verb phrase (VP) "took place at an arena that currently goes by the name TD Garden".

**Noun Phrase (NP):** The NP "WWE Super Tuesday" acts as the main subject and sets the stage for the verb phrase, grounding the sentence in the context of a wrestling entertainment event.

Verb Phrase (VP): The verb phrase "took place at an arena that currently goes by the name TD Garden" begins with the verb "took" linked to the noun "place," collectively forming the predicate "took place." This construction indicates the occurrence of an event. The phrase then extends to include a prepositional phrase (PP) "at an arena" that specifies the location. This prepositional phrase includes an additional NP "an arena," which is modified by a relative clause. This relative clause "that currently goes by the name TD Garden" adds specificity to "an arena." The clause starts with the pronoun "that," referring back to "arena," and includes a complex structure: An adverbial phrase "currently" modifies the verb "goes," indicating the timing of the action; and a verb "goes" is followed by another prepositional phrase "by the name TD Garden," where "the name TD Garden" is a nested NP specifying the current name of the arena.

By decomposing the original sentence, we can derive two simplified sub-sentences: 1.WWE Super Tuesday took place at an arena. 2.The arena currently goes by the name TD Garden.

We can pose the first part of the claim as a question using the Question() function: "Which arena the WWE Super Tuesday took place?". This question aims to identify the specific arena where the event happened.The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "{answer\_1} currently goes by the name TD Garden". This is done using the Verify() function and stored in the variable fact\_1 .Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (fact\_1) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

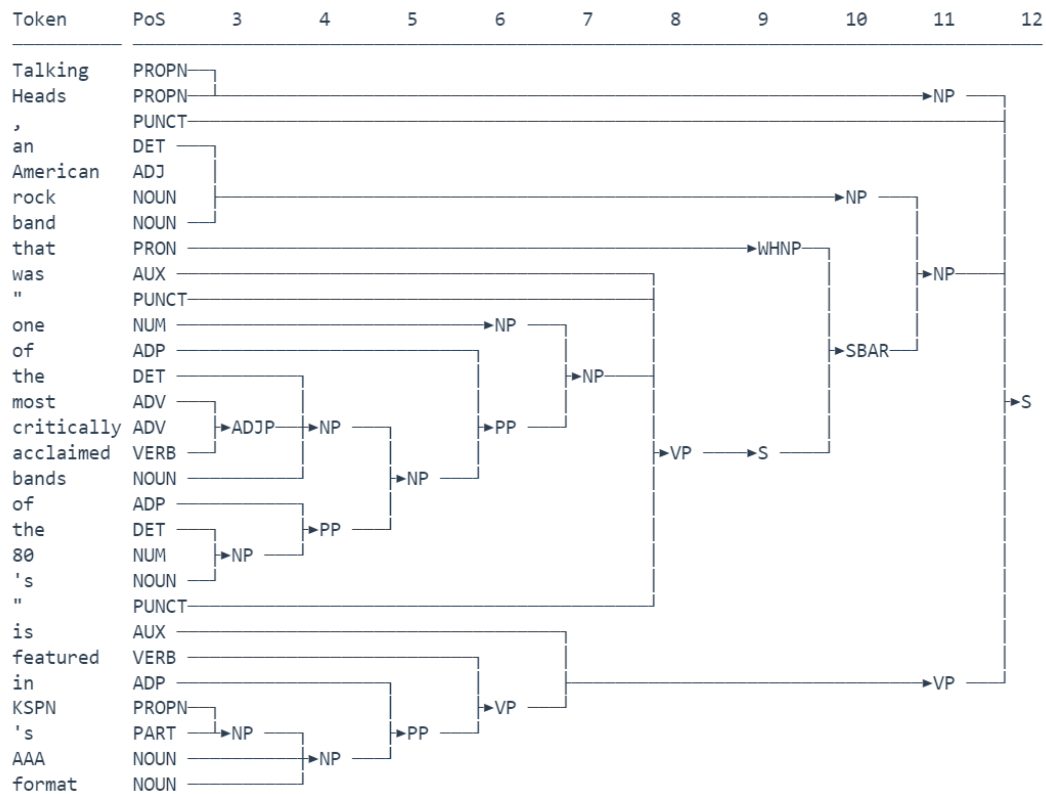
Program code:

```
answer_1 = Question("Which arena the WWE Super Tuesday took place
?")
fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden
.")
label = Predict(fact_1)
```

# The claim is that Talking Heads, an American rock band that was "one of the most critically acclaimed bands of the 80's" is featured in KSPN's AAA format.

Constituency parsing tree:

Analysis: Using a top-down approach, we first identify the major divisions within the sentence, which are divided into a complex noun phrase (NP)"Talking Heads, an American rock band that was "one of the most critically acclaimed bands of the 80's"" as the subject and a verb phrase (VP)"is featured in KSPN's AAA format" as the predicate.



**Noun Phrase (NP):**The initial NP "Talking Heads" acts as the main subject, which is a proper noun referring to the name of a band. This NP is then expanded with an additional descriptive clause that provides more information about "Talking Heads." This expansion starts with ", an American rock band," which itself is a noun phrase describing what "Talking Heads" is, further identifying the genre and nationality of the band. This clause can be further divided into 2 parts: 1. It begins with the pronoun "that," linking back to "American rock band." 2. The clause includes a verb "was," leading into another nested noun phrase "one of the most critically acclaimed bands of the 80s."

**Verb Phrase (VP):**The verb phrase following the extensive subject details begins with "is featured," indicating the current status or activity regarding the band. The verb "is featured" links to a prepositional phrase "in KSPN's AAA format."

By decomposing the original sentence, we can derive three simplified sub-sentences: 1. Talking Heads is an American rock band that was one of the most critically acclaimed bands of the 80's. 2. Talking Heads is featured in KSPN's AAA format.



Based on this information, we can transform each fact into a Verify() function. The first Verify() function checks if the "Talking Heads is an American rock band that was 'one of the most critically acclaimed bands of the 80's'", and the second Verify() function checks if "Talking Heads is featured in KSPN's AAA format". Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two Verify() functions. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

Program code:

```
fact_1 = Verify("Talking Heads is an American rock band that was '
one of the most critically acclaimed bands of the 80's'.")
fact_2 = Verify("Talking Heads is featured in KSPN's AAA format.")
label = Predict(fact_1 and fact_2)
```

```
The claim is that The model of car Trevor Bayne drives was
introduced for model year 2006. The Rookie of The Year in the 1997
CART season drives it in the NASCAR Sprint Cup.
```

Constituency parsing tree:

ewpage

Analysis: We employ a top-down approach to analyze the tree. The structure as divided into two main sentence components (S) with their respective noun phrases (NP) and verb phrases (VP).

Analysis of the First Sentence (S):

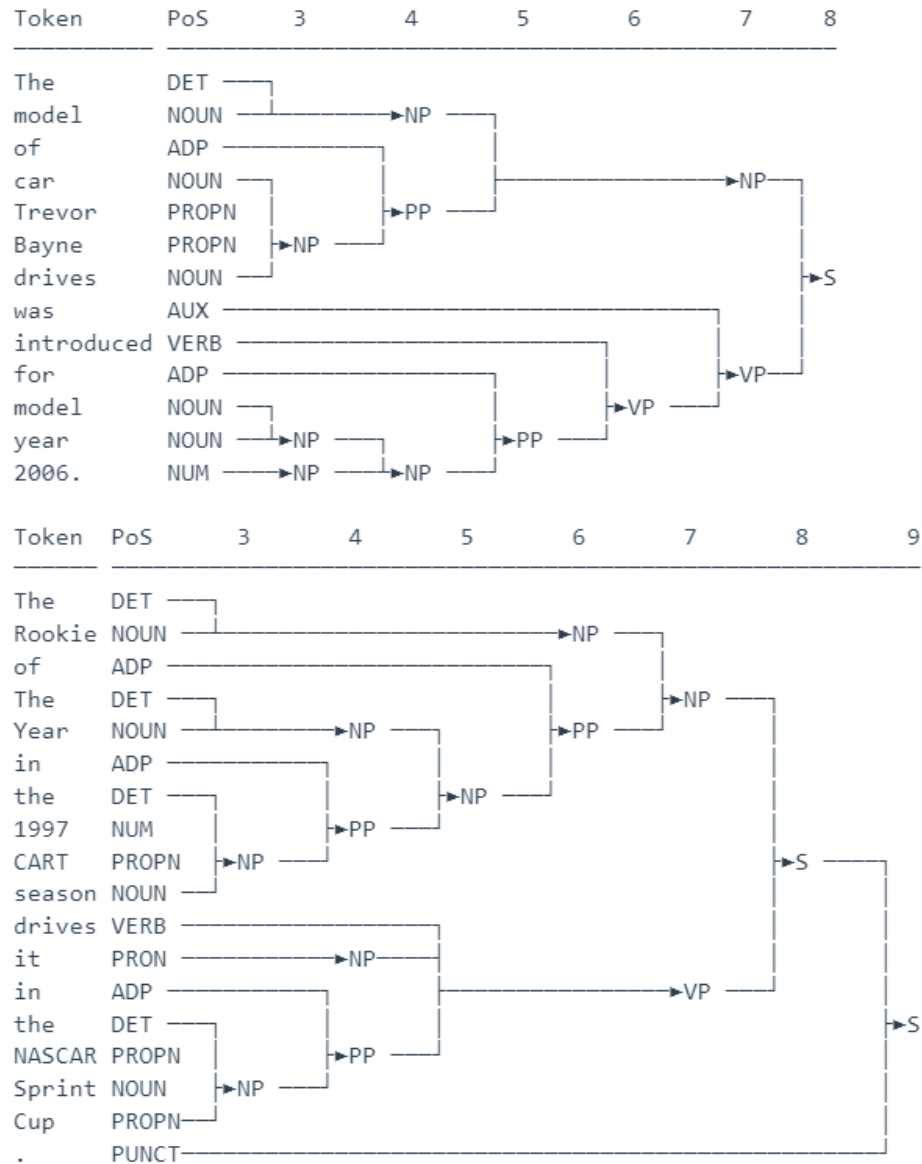
Noun Phrase (NP): The initial NP "The model of car Trevor Bayne drives" acts as the main subject of the first sentence. Starting with "The model," which is the core noun phrase, it is expanded by a prepositional phrase "of car Trevor Bayne drives," which includes a nested NP "car Trevor Bayne" modified further by the NP "Trevor Bayne," identifying the specific individual associated with the car.

Verb Phrase (VP): The verb phrase "was introduced for model year 2006" follows, describing the action and the temporal context concerning the car's model. The verb "was introduced" indicates the introduction of the car model, linked by the preposition "for" to another NP "model year 2006".

Analysis of the Second Sentence (S):

Noun Phrase (NP): The subject of the second sentence, "The Rookie of the Year in the 1997 CART season," is structured with a main NP "The Rookie of the Year," expanded by the prepositional phrase "in the 1997 CART season." This phrase is further detailed by the NPs "the 1997" and "CART season," which place the rookie title within a specific historical and sporting context.

Verb Phrase (VP): The VP "drives it in the NASCAR Sprint Cup" involves the main action "drives," referring back to the subject of the first sentence. The object "it" likely refers to the car model discussed earlier. The phrase is concluded with another prepositional phrase "in the NASCAR Sprint Cup," naming the competition where the action takes place.



By decomposing the original sentence, we can derive two simplified sub-sentences: 1. The model of car that Trevor Bayne drives was introduced in 2006. 2. The Rookie of the Year in the 1997 CART season drives that model in the NASCAR Sprint Cup.

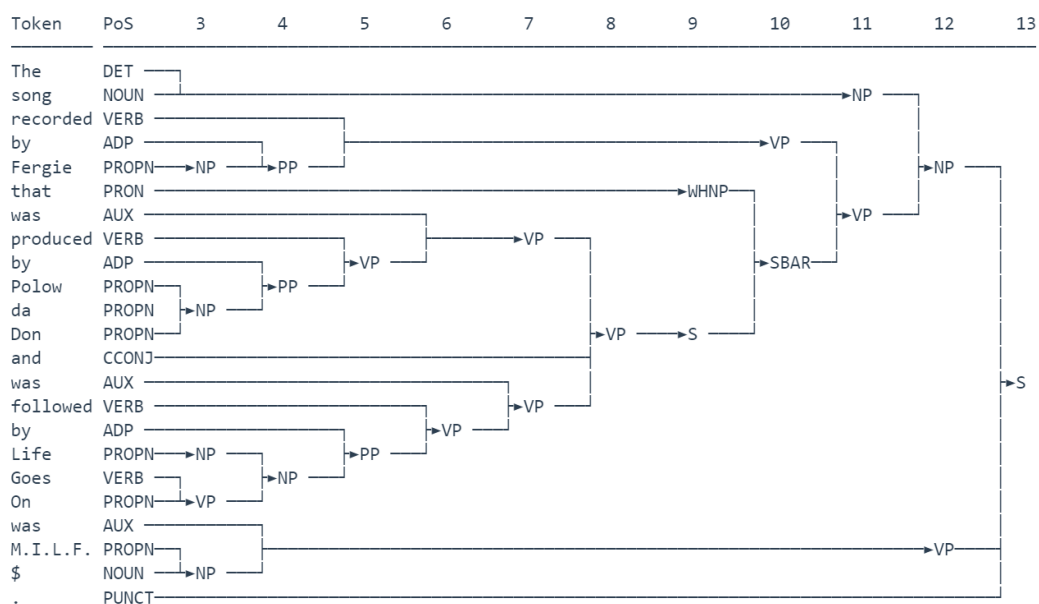
We can pose the first part of the claim as a question using the Question() function: "Which model of car is driven by Trevor Bayne?". This question aims to identify the specific model of car Trevor Bayne drives. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "{answer\_1} was introduced for model year 2006". This is done using the Verify() function and stored in the variable fact\_1. Next, we pose the second part of the claim as a question using the Question() function: "Who is the Rookie of The Year in the 1997 CART season?". This question aims to identify the specific person who is the Rookie of The Year in the 1997 CART season. The answer to this question is stored in the variable answer\_2. Then, we use these answers to verify the fact that "{answer\_2} drives {answer\_1} in the NASCAR Sprint Cup". This is done using the Verify() function and stored in the variable fact\_2. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the results of the two Verify() functions (fact\_1 and fact\_2) as inputs. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

Program code:

```
answer_1 = Question("Which model of car is driven by Trevor Bayne
?")
fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
answer_2 = Question("Who is the Rookie of The Year in the 1997
CART season?")
fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR
Sprint Cup.")
label = predict(fact_1 and fact_2)
```

# The claim is that The song recorded by Fergie that was produced by Polow da Don and was followed by Life Goes On was M.I.L.F.\$.

Constituency parsing tree:



Analysis: Using a top-down approach, the sentence is primarily divided into a noun phrase(NP)"The song recorded by Fergie that was produced by Polow da Don and was followed by Life Goes On" and a verb phrase(VP)"was M.I.L.F. \$"

Noun Phrase (NP) : "The song" serves as the main subject of the sentence. The NP can be further decompose into 3 clauses.

Relative Clause 1: Modified by "recorded by Fergie," where "by Fergie" acts as a prepositional phrase (PP) pointing to the artist involved in the recording.

Nested Clause: Within this first clause, there's another clause "that was produced by Polow da Don." Here, "that" refers back to "the song," "was produced" forms the verb phrase (VP), and "by Polow da Don" is another prepositional phrase describing the producer.

Relative Clause 2: "and was followed by 'Life Goes On'" connects with a conjunction "and." "Was followed" is the verb, and "by 'Life Goes On'" is the prepositional phrase indicating what followed the song.

Verb Phrase (VP): "M.I.L.F. \$" is the complement of "was," completing the predicate by specifying the songs title.

By decomposing the original sentence, we can derive two simplified sub-sentences:1. M.I.L.F. \$ was recorded by Fergie 2. M.I.L.F. \$ was produced by Polow da Don 3. M.I.L.F. \$ was followed by 'Life Goes On'

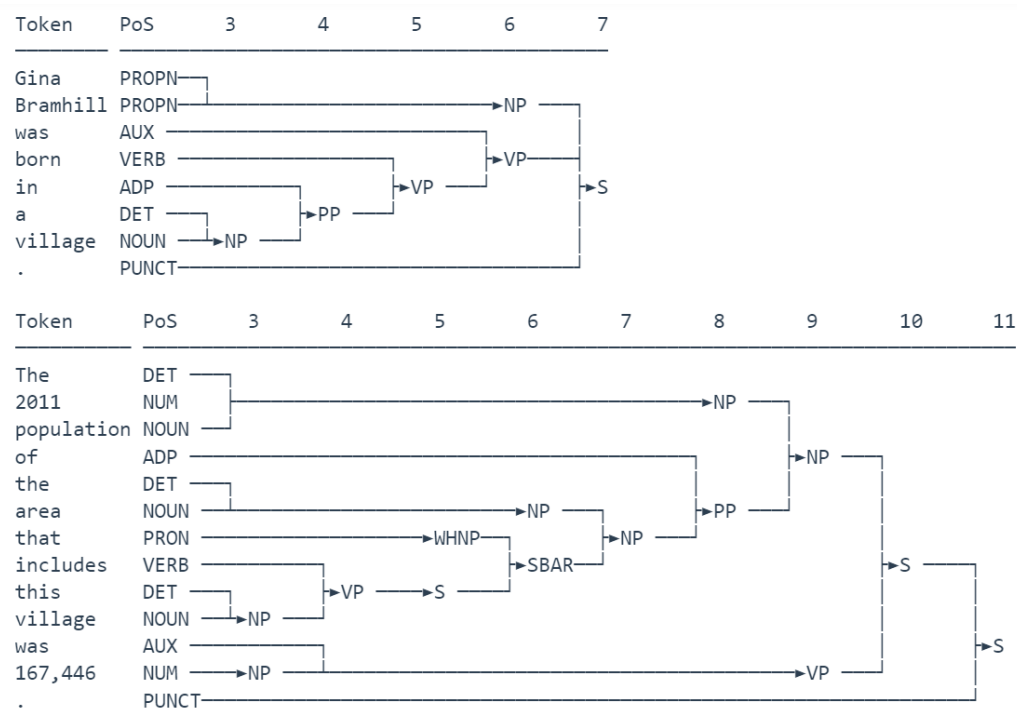
Based on this information, we can transform each fact into a Verify() function. The first Verify() function checks if the "M.I.L.F.\$ was recorded by Fergie", the second Verify() function checks if "M.I.L.F.\$ was produced by Polow da Don", and the third Verify() function checks if "M.I.L.F.\$ was followed by Life Goes On". Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the three Verify() functions. If all Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

Program code:

```
fact_1 = Verify("M.I.L.F.$ was recorded by Fergie.")
fact_2 = Verify("M.I.L.F.$ was produced by Polow da Don.")
fact_3 = Verify("M.I.L.F.$ was was followed by Life Goes On.")
label = Predict(fact_1 and fact_2 and fact_3)
```

# The claim is that Gina Bramhill was born in a village. The 2011 population of the area that includes this village was 167,446.

Constituency parsing tree:



Analysis: The claim involves two facts about Gina Bramhill and the village she was born in. The first fact is that "Gina Bramhill was born in a village" and the second fact is that "The 2011 population of the area that includes this village was 167,446". The constituency parsing tree helps us break down the sentence into its constituent parts. The tree shows that the sentence has a main branch that represents the first fact: "Gina Bramhill was born in a village". The village is further described in the second fact: "The 2011 population of the area that includes this village was 167,446". From the parse tree, we can identify the subject of the first fact: "Gina Bramhill" (NP), the action itself: "was born" (VERB), and the location of the action: "in a village" (PP).

We can pose the first part of the claim as a question using the Question() function: "Which village was Gina Bramhill born in?". This question aims to identify the specific village where Gina Bramhill was born. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the second fact that "The 2011 population of the area that includes {answer\_1} was 167,446". This is done using the Verify() function and stored in the variable fact\_1. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (fact\_1) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

Program code:

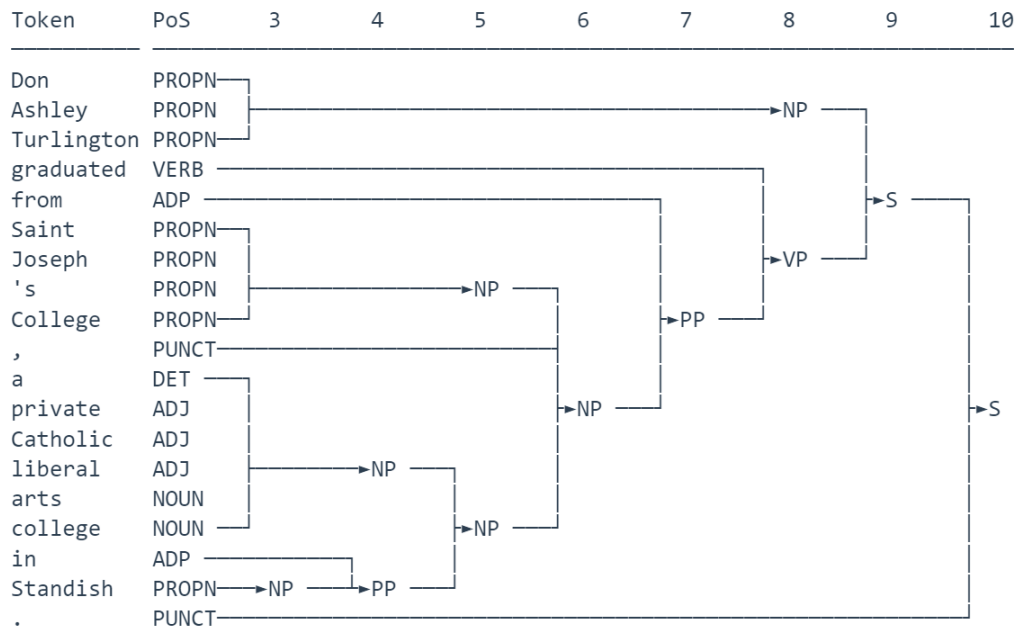
```

answer_1 = Question("Which village was Gina Bramhill born in?")
fact_1 = Verify(f"The 2011 population of the area that includes {
answer_1} was 167,446.")
label = Predict(fact_1)

```

# The claim is that Don Ashley Turlington graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish.  
 Constituency parsing tree:

ewpage



**Analysis:** We use a top-down approach to examine the structure as outlined by the tree. The sentence is organized around a central noun phrase (NP) "Don Ashley Turlington" and a verb phrase (VP) "graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish".

**Noun Phrase (NP) "Don Ashley Turlington":** This NP serves as the subject of the sentence, identifying the individual involved.

**Verb Phrase (VP) "graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish":** The action "graduated" is the main verb. It links the subject to the institution from which they graduated, which is a Prepositional Phrase (PP) "from Saint Joseph's College, a private Catholic liberal arts college in Standish". Appositive NP "a private Catholic liberal arts college in Standish" provides additional descriptive information about Saint Joseph's College.

From the full sentence structure, simplified sub-sentences can be derived: 1. Don Ashley Turlington graduated from Saint Joseph's College. 2. Saint Joseph's College is a private Catholic liberal arts college in Standish.

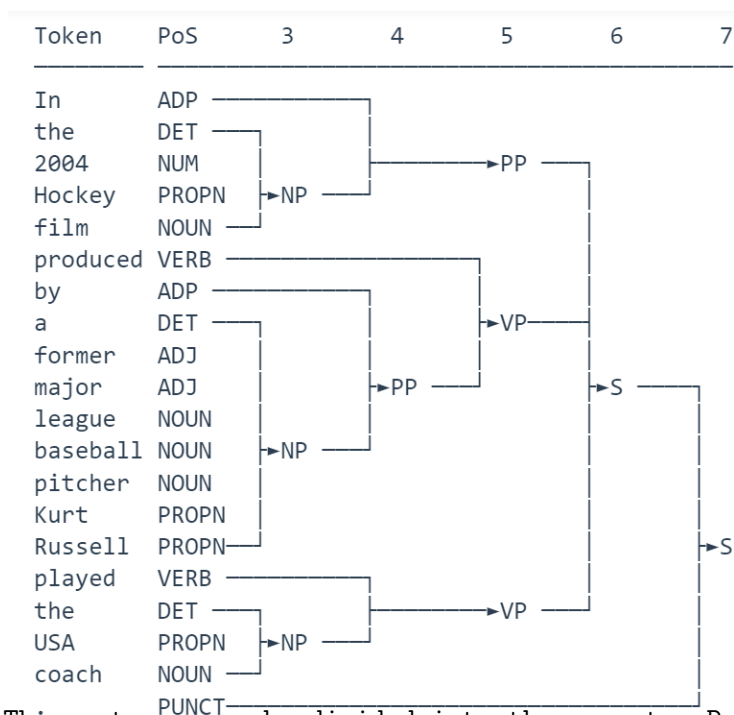
Based on this information, we can create two `Verify()` functions. The first `Verify()` function checks if "Saint Joseph's College is a private Catholic liberal arts college in Standish". This fact is derived from the appositive phrase that describes the college. The second `Verify()` function checks if "Don Ashley Turlington graduated from Saint Joseph's College". This fact is directly stated in the main clause of the sentence. Finally, the `Predict()` function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two `Verify()` functions. If both `Verify()` functions return true, then the `Predict()` function will also return true, meaning the overall claim is true.

Program code:

```
fact_1 = Verify("Saint Joseph's College is a private Catholic
liberal arts college in Standish.")
fact_2 = Verify(f"Don Ashley Turlington graduated from Saint
Joseph's College.")
label = Predict(fact_1 and fact_2)
```

# The claim is that In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach.

Constituency parsing tree:



Analysis: The sentence can be divided into three parts: Prepositional Phrase (PP) 'In the 2004 Hockey film', Verb Phrase (VP) 'produced by a former major league baseball pitcher', and main Verb (VP) 'played the USA coach'.

Prepositional Phrase (PP): The sentence starts with a prepositional phrase "In," which sets the temporal and thematic context. Within this phrase, "the 2004 Hockey film" acts as a noun phrase (NP) that specifies which film is being discussed.

Verb Phrase (VP): The action "produced" links the film to its producer . This verb phrase includes a complex prepositional phrase "by a former major league baseball pitcher," providing detailed background about the producer.

Main Verb (VP): The main clause of the sentence is "Kurt Russell played the USA coach." Here, "played" is the main verb indicating the action performed by Kurt Russell. The object "the USA coach" is a noun phrase describing the role that Kurt Russell assumed in the film.

From the full sentence structure, simplified sub-sentences can be derived: 1. In the 2004 Hockey film, Kurt Russell played the USA coach. 2. The film was produced by Kurt Russell, a former major league baseball pitcher.

To extract the information needed: 1. We use the Question() function to ask about the specific film: "Which 2004 Hockey film was produced a former major league baseball pitcher?". This question aims to identify the film produced by the former pitcher. The answer to this question is stored in the variable answer\_1. 2. Next, we use this answer to verify whether Kurt Russell played the USA coach in this specific film, forming the Verify() function: "Kurt Russell played the USA coach in the film {answer\_1}." This checks the accuracy of Kurt Russells role in the identified film. Finally, we use the Predict() function to predict the veracity label of the overall claim based on the result of the Verify() function. If the Verify() function returns true, then the Predict () function will also return true, meaning the overall claim is true.

Program code:

```
answer_1 = Question("Which 2004 Hockey film was produced a former
major league baseball pitcher?")
fact_1 = Verify("Kurt Russell played the USA coach in the film {
answer_1}.")
label = Predict(fact_1)
```

# The claim is that Jack McFarland is the best known role of the host of the 64th Annual Tony Awards.

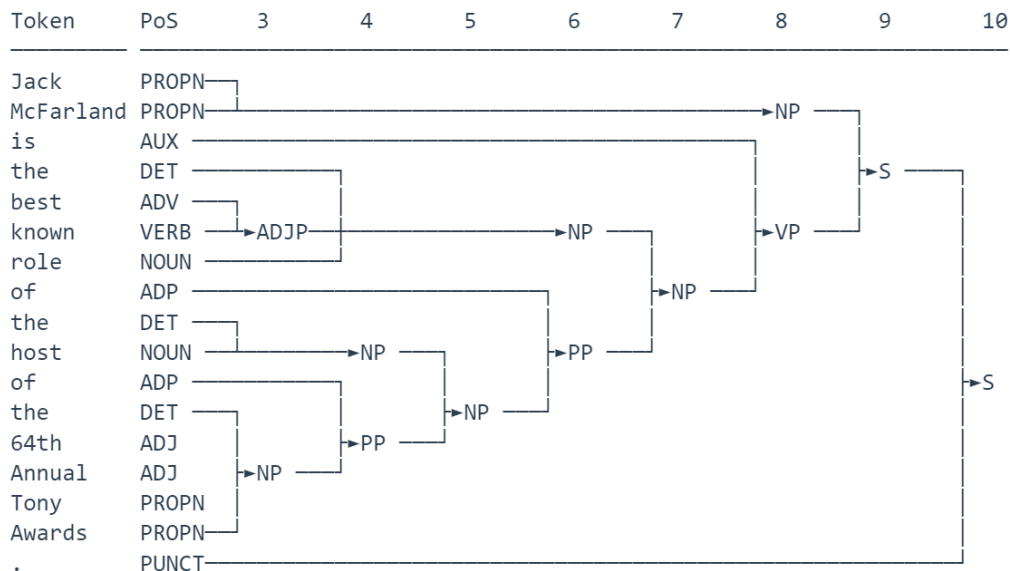
Constituency parsing tree:

Analysis: We'll use a top-down approach to break down the structure based on its components. The sentence is organized into two main components: anoun phrase (NP)"Jack McFarland" identifying the subject of the sentence, and a verb phrase (VP) "is the best known role of the host of the 64th Annual Tony Awards.", provides additional information about the subject.

Noun Phrase (NP): This NP serves as the main subject of the sentence.

Verb Phrase (VP): The linking verb "is" connects the subject "Jack McFarland" to the descriptive clause that follows. The remaining clause could be further divided into NP "the best known role" and PP "of the host of the 64th Annual Tony Awards".





From the full sentence structure, simplified sub-sentences can be derived: 1. Jack McFarland served as the host of the 64th Annual Tony Awards. 2. Jack McFarland is best known for his role.

We can pose the first part of the claim as a question using the Question() function: "Who are the hosts of the 64th Annual Tony Awards?" This question aims to identify the specific host of the event. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "Jack McFarland is the best known role of {answer\_1}." This is done using the Verify() function and stored in the variable fact\_1. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (fact\_1) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

Program code:

```
answer_1 = Question("Who are the hosts of the 64th Annual Tony Awards?")
fact_1 = Verify(f"Jack McFarland is the best known role of {answer_1}.")
label = Predict(fact_1)
```

# The claim is that [[CLAIM]]

Constituency parsing tree:

[[TREE]]

Analysis:

## A.5 Chain-of-Thought with Supervision(CoT-S)

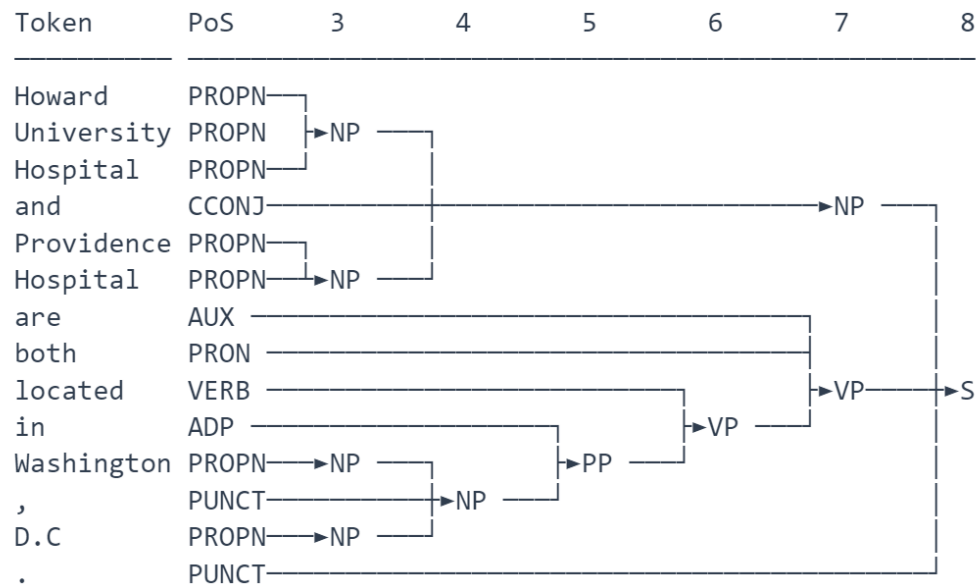
```
[
 {
 "role": "system",
```

"content": "You have strong ability of reasoning. Given a claim and its constituency tree, generate analysis about how to decompose the claim based on the tree, and a pseudo program that describes the reasoning steps required to verify the claim step-by-step. The constituency tree is to help you understand and decompose. You can only call three functions in the program: 1. Question() to answer a question; 2. Verify() to verify a simple claim; 3. Predict() to predict the veracity label, the only logical Operators can be used is 'and'. Your reasoning steps should start with 'Analysis:', and the program code you generated should start with 'Program Code:'."

```
},
{
```

```
 "role": "user",
```

```
 "content": "The claim is that Howard University Hospital and Providence Hospital are both located in Washington, D.C.. The constituency parsing tree is:
```



```
},
{
```

```
 "role": "assistant",
```

```

"content": "Analysis: To verify the claim "Howard University
Hospital and Providence Hospital are both located in Washington, D
.C.", we need to break it down into simpler components based on
the constituency tree. The tree shows that the claim consists of
two main entities (Howard University Hospital and Providence
Hospital) and a location (Washington, D.C.). 1. Identify the
first entity: "Howard University Hospital". 2. Identify the
second entity: "Providence Hospital". 3. Identify the location: "
Washington, D.C.". 4. Verify if "Howard University Hospital" is
located in "Washington, D.C.". 5. Verify if "Providence Hospital"
is located in "Washington, D.C.". 6. Combine the results of the
two verifications to determine if both hospitals are located in
Washington, D.C. Program Code: # Step 1: Verify if Howard
University Hospital is located in Washington, D.C. verify_howard
= Verify("Howard University Hospital is located in Washington, D.C
.") # Step 2: Verify if Providence Hospital is located in
Washington, D.C. verify_providence = Verify("Providence Hospital
is located in Washington, D.C.") # Step 3: Combine the results
result = Predict(verify_howard and verify_providence) "
},
{
 "role": "user",
 "content": "The suggested analysis is: At the top, the sentence
is divided into a large noun phrase (NP) and a verb phrase (VP).
The noun phrase itself is a compound structure, made up of two
separate entities: "Howard University Hospital" and "Providence
Hospital." These entities are connected by the conjunction "and,"
indicating that the action described in the verb phrase applies
equally to both entities. Moving to the verb phrase, it consists
of the auxiliary verb "are," the adverb "both," the main verb "
located," and a prepositional phrase "in Washington, D.C." This
structure tells us that the state of being located is shared by
both subjects of the compound noun phrase. The prepositional
phrase at the end of the sentence specifies the location,
indicating that both hospitals are situated in Washington, D.C.
In breaking down this sentence, we observe how each part of the
structure contributes to the overall meaning. The conjunction in
the noun phrase effectively binds the two hospitals into a shared
circumstance, while the components of the verb phrase distribute
this circumstance across both subjects equally. Thus, using the
constituency tree helps to visually and logically segment the
sentence into digestible parts without altering the intended
meaning. This detailed analysis shows how both hospitals, though
mentioned separately, are unified in their description,
emphasizing their collective presence in the same geographical
location.

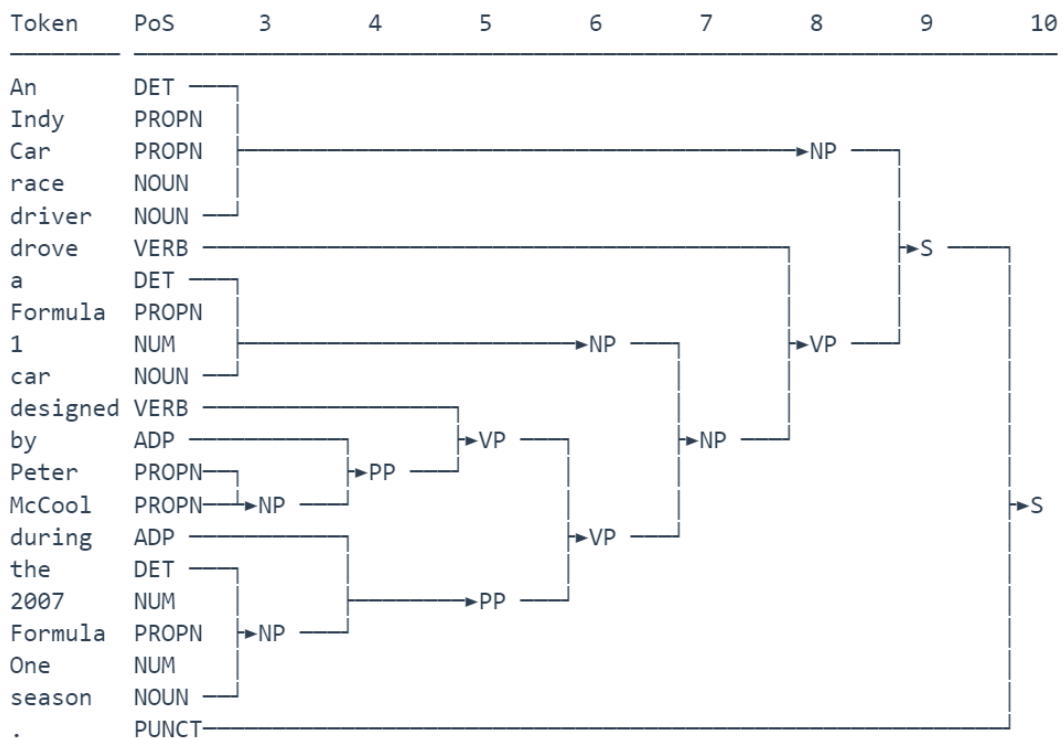
```

From this analysis, we can derive two simpler sub-sentences: 1. Howard University Hospital is located in Washington, D.C. 2. Providence Hospital is also located in Washington, D.C. Based on this information, we can transform each fact into a Verify() function. The first Verify() function checks if the "Howard University Hospital is located in Washington, D.C." and the second Verify() function checks if "Providence Hospital is located in Washington, D.C.". Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two Verify() functions. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true., and the correct answer is: fact\_1 = Verify("Howard University Hospital is located in Washington, D.C.") fact\_2 = Verify("Providence Hospital is located in Washington, D.C.") label = Predict(fact\_1 and fact\_2). Carefully examine the differences between your answers and the standard ones. Learn from the correct decomposing approaches and formats, and apply these insights to your next claim."

```

},
{
 "role": "user",
 "content": "The claim is that An IndyCar race driver drove a
 Formula 1 car designed by Peter McCool during the 2007 Formula One
 season. The constituency parsing tree is:

```



```

"
},
{

```

```

"role": "assistant",
"content": "Analysis: To verify the claim \"An IndyCar race driver
drove a Formula 1 car designed by Peter McCool during the 2007
Formula One season,\" we need to break it down into simpler
components based on the constituency tree. The tree shows that the
claim consists of several key elements: 1. The subject: \"An
IndyCar race driver\". 2. The action: \"drove\". 3. The object: \"a
Formula 1 car designed by Peter McCool\". 4. The time frame: \"
during the 2007 Formula One season\". From this analysis, we can
derive simpler sub-sentences: 1. An IndyCar race driver drove a
Formula 1 car. 2. The Formula 1 car was designed by Peter McCool.
3. The event occurred during the 2007 Formula One season. We
can transform each fact into a Verify() function. The first Verify
() function checks if \"An IndyCar race driver drove a Formula 1
car.\" The second Verify() function checks if \"The Formula 1 car
was designed by Peter McCool.\" The third Verify() function checks
if \"The event occurred during the 2007 Formula One season.\"
Finally, the Predict() function is used to predict the veracity
label of the overall claim. The logical operator 'and' is used to
combine the results of the three Verify() functions. If all Verify
() functions return true, then the Predict() function will also
return true, meaning the overall claim is true.
Program Code: # Step 1: Verify if an IndyCar race driver drove a
Formula 1 car. fact_1 = Verify(\"An IndyCar race driver drove a
Formula 1 car.\") # Step 2: Verify if the Formula 1 car was
designed by Peter McCool. fact_2 = Verify(\"The Formula 1 car was
designed by Peter McCool.\") # Step 3: Verify if the event
occurred during the 2007 Formula One season. fact_3 = Verify(\"The
event occurred during the 2007 Formula One season.\") # Step 4:
Combine the results label = Predict(fact_1 and fact_2 and fact_3
)\"
},
{
"role": "user",
"content": "The suggested analysis is:
We use a top-down approach to analyze this sentence. We start by
identifying the highest node in the parsing tree, which
categorizes the entire sentence into two primary components: a
noun phrase (NP) and a verb phrase (VP).
Noun Phrase (NP):At the top level, the NP \"An IndyCar race driver\"
is selected. This phrase is identified as the subject of the
sentence. The NP combines a determiner \"An\" with a compound noun \"
IndyCar race driver\". This NP sets the stage by identifying who is
performing the action, grounding the sentence in the world of
IndyCar racing.

```

Verb Phrase (VP): Following the subject, we examine the VP "drove a Formula 1 car designed by Peter McCool during the 2007 Formula One season." This complex VP begins with the verb "drove," which is the main action connecting the subject to the direct object "a Formula 1 car." This object itself extends into a more detailed description through additional nested structures: 1. The phrase "designed by Peter McCool" is an embedded clause that modifies "a Formula 1 car," specifying that the car was designed by Peter McCool. Here, "designed" acts as a verb in a smaller verb phrase, which includes a prepositional phrase "by Peter McCool." "Peter McCool" serves as a noun phrase within this prepositional phrase, attributing the design to him. 2. The final part of the VP, "during the 2007 Formula One season," provides the temporal context. This is structured as a prepositional phrase containing "the 2007 Formula One season," which acts as a noun phrase detailing when the action took place.

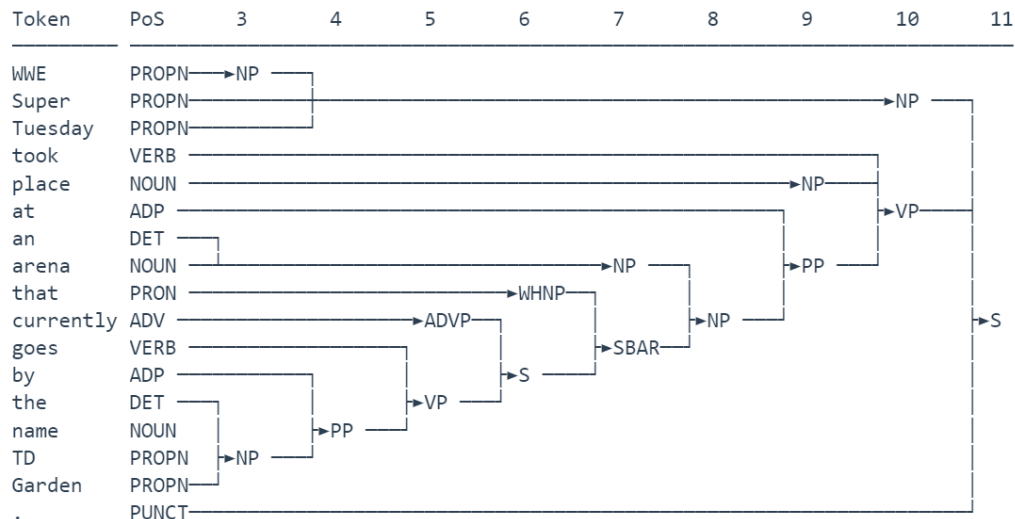
By decomposing the original sentence, we can derive two simplified sub-sentences that preserve the meaning but clarify the structure: 1. An IndyCar race driver drove a Formula 1 car. 2. The car was designed by Peter McCool during the 2007 Formula One season.

We can pose the second part of the claim as a question using the Question() function: "Which Formula 1 car was designed by Peter McCool during the 2007 Formula One season?". This question aims to identify the specific car that the IndyCar race driver drove. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "An IndyCar race driver drove the car {answer\_1}". This is done using the Verify() function and stored in the variable fact\_1. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (fact\_1) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true., and the correct answer is:

```
answer_1 = Question("Which Formula 1 car was designed by Peter
 McCool during the 2007 Formula One season?")
fact_1 = Verify(f"An IndyCar race driver drove the car {answer_1}.")
label = Predict(fact_1).
```

Carefully examine the differences between your answers and the standard ones. Learn from the correct decomposing approaches and formats, and apply these insights to your next claim."

```
},
{
 "role": "user",
 "content": "The claim is that WWE Super Tuesday took place at an arena that currently goes by the name TD Garden. The constituency parsing tree is:
"
},
{
 "role": "assistant",
```



"content": "Analysis: To verify the claim "WWE Super Tuesday took place at an arena that currently goes by the name TD Garden," we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of several key elements: 1. The event: "WWE Super Tuesday". 2. The action: "took place". 3. The location: "at an arena". 4. The description of the arena: "that currently goes by the name TD Garden". From this analysis, we can derive simpler sub-sentences: 1. WWE Super Tuesday took place at an arena. 2. The arena currently goes by the name TD Garden. We can pose the second part of the claim as a question using the Question() function: "Which arena currently goes by the name TD Garden?". This question aims to identify the specific arena where WWE Super Tuesday took place. The answer to this question is stored in the variable `answer\_1`. Then, we use this answer to verify the fact that "WWE Super Tuesday took place at the arena {answer\_1}". This is done using the Verify() function and stored in the variable `fact\_1`. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (`fact\_1`) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true. Program Code: # Step 1: Ask which arena currently goes by the name TD Garden. answer\_1 = Question("Which arena currently goes by the name TD Garden?") # Step 2: Verify if WWE Super Tuesday took place at the identified arena. fact\_1 = Verify(f"WWE Super Tuesday took place at the arena {answer\_1}.") # Step 3: Combine the results label = Predict(fact\_1)"

```

},
{
 "role": "user",

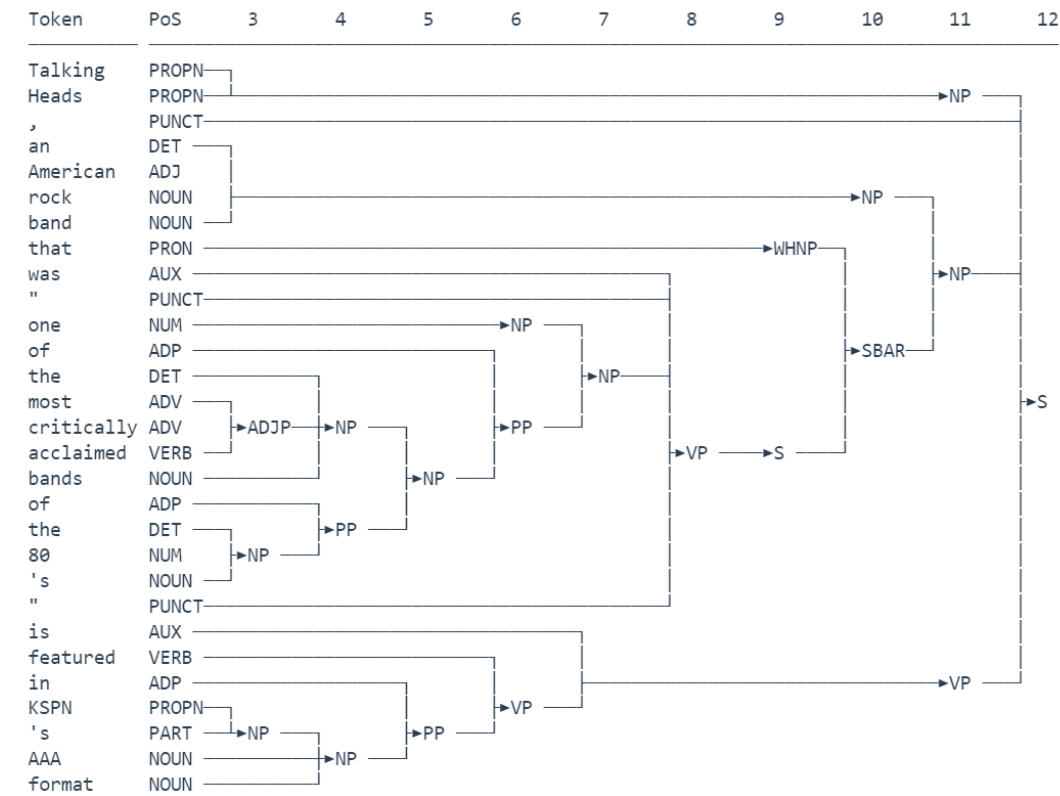
```

```

"content": "The suggested analysis is: We use a top-down approach
to analyze this sentence. The sentence's structure is initially
split into a noun phrase(NP) "WWE Super Tuesday" and a verb phrase
(VP) "took place at an arena that currently goes by the name TD
Garden". Noun Phrase (NP): The NP "WWE Super Tuesday" acts as
the main subject and sets the stage for the verb phrase, grounding
the sentence in the context of a wrestling entertainment event.
Verb Phrase (VP): The verb phrase "took place at an arena that
currently goes by the name TD Garden" begins with the verb "took"
linked to the noun "place," collectively forming the predicate "
took place." This construction indicates the occurrence of an
event. The phrase then extends to include a prepositional phrase (
PP) "at an arena" that specifies the location. This prepositional
phrase includes an additional NP "an arena," which is modified by
a relative clause. This relative clause "that currently goes by
the name TD Garden" adds specificity to "an arena." The clause
starts with the pronoun "that," referring back to "arena," and
includes a complex structure: An adverbial phrase "currently"
modifies the verb "goes," indicating the timing of the action; and
a verb "goes" is followed by another prepositional phrase "by the
name TD Garden," where "the name TD Garden" is a nested NP
specifying the current name of the arena. By decomposing the
original sentence, we can derive two simplified sub-sentences: 1.
WWE Super Tuesday took place at an arena. 2.The arena currently
goes by the name TD Garden. We can pose the first part of the
claim as a question using the Question() function: "Which arena
the WWE Super Tuesday took place?". This question aims to identify
the specific arena where the event happened.The answer to this
question is stored in the variable answer_1. Then, we use this
answer to verify the fact that "{answer_1} currently goes by the
name TD Garden". This is done using the Verify() function and
stored in the variable fact_1.Finally, we use the Predict()
function to predict the veracity label of the overall claim. The
Predict() function uses the result of the Verify() function (
fact_1) as an input. If the Verify() function returns true, then
the Predict() function will also return true, meaning the overall
claim is true., and the correct answer is: answer_1 =
Question("Which arena the WWE Super Tuesday took place?")
fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden
.") label = Predict(fact_1). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
"role": "user",
"content": "The claim is that Talking Heads, an American rock
band that was "one of the most critically acclaimed bands of the
80's" is featured in KSPN's AAA format.. The constituency parsing
tree is:

```





"

},  
{

"role": "assistant",  
"content": "

Analysis: To verify the claim "Talking Heads, an American rock band that was 'one of the most critically acclaimed bands of the 80's' is featured in KSPN's AAA format," we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of several key elements: 1. The subject: "Talking Heads, an American rock band that was 'one of the most critically acclaimed bands of the 80's'". 2. The action: "is featured". 3. The location: "in KSPN's AAA format". From this analysis, we can derive simpler sub-sentences: 1. Talking Heads is an American rock band. 2. Talking Heads was one of the most critically acclaimed bands of the 80's. 3. Talking Heads is featured in KSPN's AAA format.

We can pose the first part of the claim as a question using the Question() function: "Which American rock band was one of the most critically acclaimed bands of the 80's?". This question aims to identify the specific band that fits this description. The answer to this question is stored in the variable `answer\_1`. Then, we use this answer to verify the fact that "{answer\_1} is featured in KSPN's AAA format". This is done using the Verify() function and stored in the variable `fact\_1`. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (`fact\_1`) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

```

Program Code: # Step 1: Ask which American rock band was one of
the most critically acclaimed bands of the 80's. answer_1 =
Question("Which American rock band was one of the most critically
acclaimed bands of the 80's?") # Step 2: Verify if the
identified band is featured in KSPN's AAA format. fact_1 = Verify
(f"{answer_1} is featured in KSPN's AAA format.") # Step 3:
Combine the results label = Predict(fact_1)"
},
{
 "role": "user",
 "content": "The suggested analysis is: Using a top-down approach,
we first identify the major divisions within the sentence, which
are divided into a complex noun phrase (NP)"Talking Heads, an
American rock band that was "one of the most critically acclaimed
bands of the 80's"" as the subject and a verb phrase (VP)"is
featured in KSPN's AAA format" as the predicate. Noun Phrase (NP)
:The initial NP "Talking Heads" acts as the main subject, which is
a proper noun referring to the name of a band. This NP is then
expanded with an additional descriptive clause that provides more
information about "Talking Heads." This expansion starts with ",
an American rock band," which itself is a noun phrase describing
what "Talking Heads" is, further identifying the genre and
nationality of the band. This clause canbe further divided into 2
parts: 1. It begins with the pronoun "that," linking back to "
American rock band." 2. The clause includes a verb "was," leading
into another nested noun phrase "one of the most critically
acclaimed bands of the 80s." Verb Phrase (VP):The verb phrase
following the extensive subject details begins with "is featured,"
indicating the current status or activity regarding the band. The
verb "is featured" links to a prepositional phrase "in KSPN's AAA
format."

```

By decomposing the original sentence, we can derive three simplified sub-sentences: 1. Talking Heads is an American rock band that was 'one of the most critically acclaimed bands of the 80's'. 2. Talking Heads is featured in KSPN's AAA format. Based on this information, we can transform each fact into a Verify() function. The first Verify() function checks if the "Talking Heads is an American rock band that was 'one of the most critically acclaimed bands of the 80's'", and the second Verify() function checks if "Talking Heads is featured in KSPN's AAA format". Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two Verify() functions. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

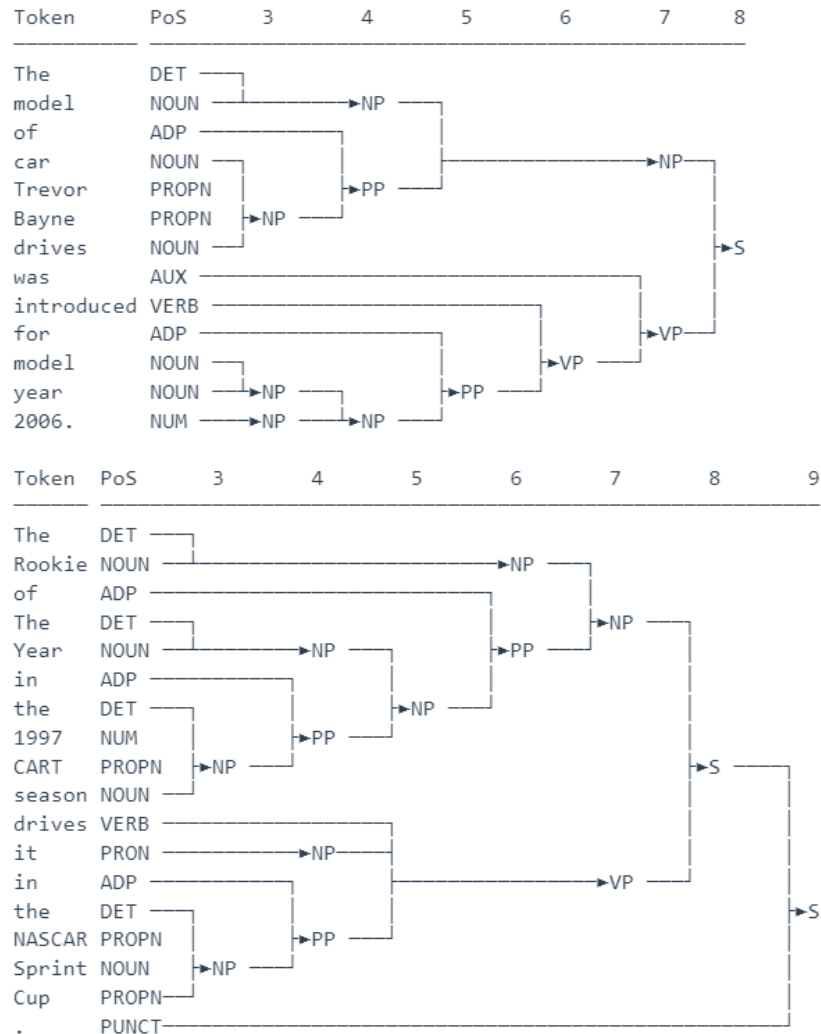
the correct answer is: fact\_1 = Verify("Talking Heads is an American rock band that was 'one of the most critically acclaimed bands of the 80's'.") fact\_2 = Verify("Talking Heads is featured in KSPN's AAA format.") label = Predict(fact\_1 and fact\_2). Carefully examine the differences between your answers and the standard ones. Learn from the correct decomposing approaches and formats, and apply these insights to your next claim."

},

{

"role": "user",

"content": "The claim is that The model of car Trevor Bayne drives was introduced for model year 2006. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup.. The constituency parsing tree is:



```

"
},
{
 "role": "assistant",
 "content": "Analysis: To verify the claim \"The model of car Trevor Bayne drives was introduced for model year 2006. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup,\" we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of two main parts: 1. The model of car Trevor Bayne drives was introduced for model year 2006. 2. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup. From this analysis, we can derive simpler sub-sentences: 1. The model of car Trevor Bayne drives was introduced for model year 2006. 2. The Rookie of The Year in the 1997 CART season drives the same model in the NASCAR Sprint Cup."

```

We can transform each fact into a Verify() function. The first Verify() function checks if "The model of car Trevor Bayne drives was introduced for model year 2006." The second Verify() function checks if "The Rookie of The Year in the 1997 CART season drives the same model in the NASCAR Sprint Cup." Finally, the Predict() function is used to predict the veracity label of the overall claim. The logical operator 'and' is used to combine the results of the two Verify() functions. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

```

Program Code: # Step 1: Verify if the model of car Trevor Bayne
drives was introduced for model year 2006. fact_1 = Verify("The
model of car Trevor Bayne drives was introduced for model year
2006.") # Step 2: Verify if the Rookie of The Year in the 1997
CART season drives the same model in the NASCAR Sprint Cup.
fact_2 = Verify("The Rookie of The Year in the 1997 CART season
drives the same model in the NASCAR Sprint Cup.") # Step 3:
Combine the results label = Predict(fact_1 and fact_2)"
},
{
 "role": "user",
 "content": "The suggested analysis is: We employ a top-down
approach to analyze the tree. The structure as divided into two
main sentence components (S) with their respective noun phrases (
NP) and verb phrases (VP). Analysis of the First Sentence (S):
Noun Phrase (NP): The initial NP "The model of car Trevor Bayne
drives" acts as the main subject of the first sentence. Starting
with "The model," which is the core noun phrase, it is expanded by
a prepositional phrase "of car Trevor Bayne drives," which
includes a nested NP "car Trevor Bayne" modified further by the NP
"Trevor Bayne," identifying the specific individual associated
with the car. Verb Phrase (VP): The verb phrase "was introduced
for model year 2006" follows, describing the action and the
temporal context concerning the car's model. The verb "was
introduced" indicates the introduction of the car model, linked by
the preposition "for" to another NP "model year 2006". Analysis
of the Second Sentence (S): Noun Phrase (NP): The subject of the
second sentence, "The Rookie of the Year in the 1997 CART season,"
is structured with a main NP "The Rookie of the Year," expanded
by the prepositional phrase "in the 1997 CART season." This phrase
is further detailed by the NPs "the 1997" and "CART season,"
which place the rookie title within a specific historical and
sporting context. Verb Phrase (VP): The VP "drives it in the
NASCAR Sprint Cup" involves the main action "drives," referring
back to the subject of the first sentence. The object "it" likely
refers to the car model discussed earlier. The phrase is concluded
with another prepositional phrase "in the NASCAR Sprint Cup,"
naming the competition where the action takes place.

```

By decomposing the original sentence, we can derive two simplified sub-sentences: 1. The model of car that Trevor Bayne drives was introduced in 2006. 2. The Rookie of the Year in the 1997 CART season drives that model in the NASCAR Sprint Cup. We can pose the first part of the claim as a question using the Question() function: "Which model of car is driven by Trevor Bayne?". This question aims to identify the specific model of car Trevor Bayne drives. The answer to this question is stored in the variable answer\_1. Then, we use this answer to verify the fact that "{answer\_1} was introduced for model year 2006". This is done using the Verify() function and stored in the variable fact\_1. Next, we pose the second part of the claim as a question using the Question() function: "Who is the Rookie of The Year in the 1997 CART season?". This question aims to identify the specific person who is the Rookie of The Year in the 1997 CART season. The answer to this question is stored in the variable answer\_2. Then, we use these answers to verify the fact that "{answer\_2} drives {answer\_1} in the NASCAR Sprint Cup". This is done using the Verify() function and stored in the variable fact\_2. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the results of the two Verify() functions (fact\_1 and fact\_2) as inputs. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

```

the correct answer is: answer_1 = Question("Which model of
car is driven by Trevor Bayne?") fact_1 = Verify(f"{answer_1}
was introduced for model year 2006.") answer_2 = Question("
Who is the Rookie of The Year in the 1997 CART season?")
fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR
Sprint Cup.") label = predict(fact_1 and fact_2).

```

Carefully examine the differences between your answers and the standard ones. Learn from the correct decomposing approaches and formats, and apply these insights to your next claim."

```

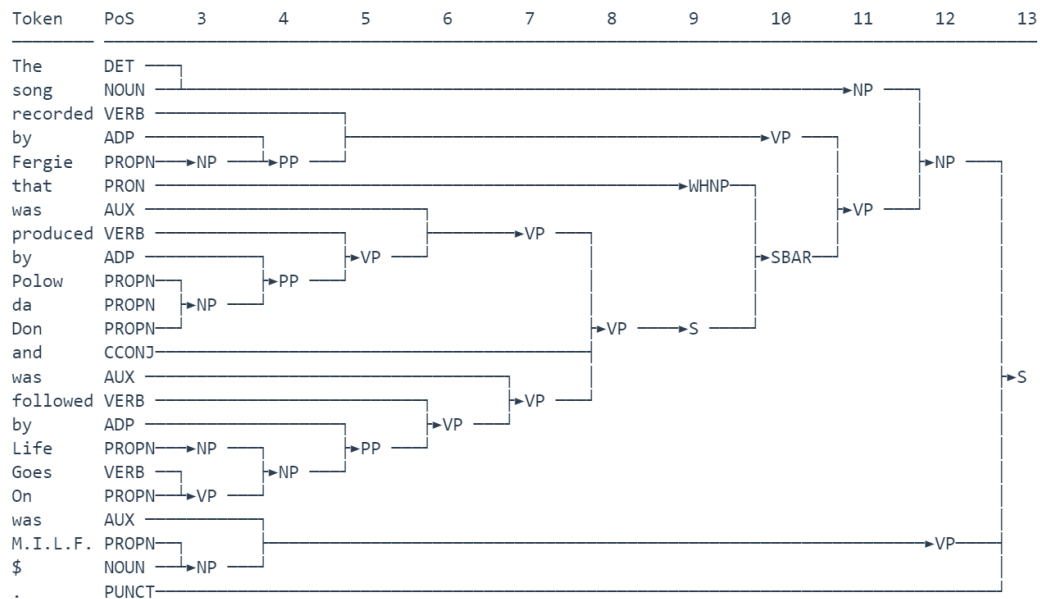
},
{
 "role": "user",
 "content": "The claim is that The song recorded by Fergie that
was produced by Polow da Don and was followed by Life Goes On was
M.I.L.F.$.. The constituency parsing tree is:

```

```

},
{
 "role": "assistant",

```



"content": "Analysis: To verify the claim "The song recorded by Fergie that was produced by Polow da Don and was followed by Life Goes On was M.I.L.F.\$.", we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of several key elements: 1. The subject: "The song recorded by Fergie". 2. The additional information about the song: "that was produced by Polow da Don and was followed by Life Goes On". 3. The identification of the song: "was M.I.L.F.\$".

From this analysis, we can derive simpler sub-sentences: 1. The song recorded by Fergie was produced by Polow da Don. 2. The song recorded by Fergie was followed by Life Goes On. 3. The song recorded by Fergie was M.I.L.F.\$.

We can pose the first part of the claim as a question using the Question() function: "Which song recorded by Fergie was produced by Polow da Don?". This question aims to identify the specific song recorded by Fergie that was produced by Polow da Don. The answer to this question is stored in the variable `answer\_1`. Then, we use this answer to verify the fact that "{answer\_1} was followed by Life Goes On". This is done using the Verify() function and stored in the variable `fact\_1`. Finally, we use the Verify() function to check if "{answer\_1} was M.I.L.F.\$". This is done using the Verify() function and stored in the variable `fact\_2`. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the results of the two Verify() functions (`fact\_1` and `fact\_2`) as inputs. If both Verify() functions return true, then the Predict() function will also return true, meaning the overall claim is true.

Program Code:

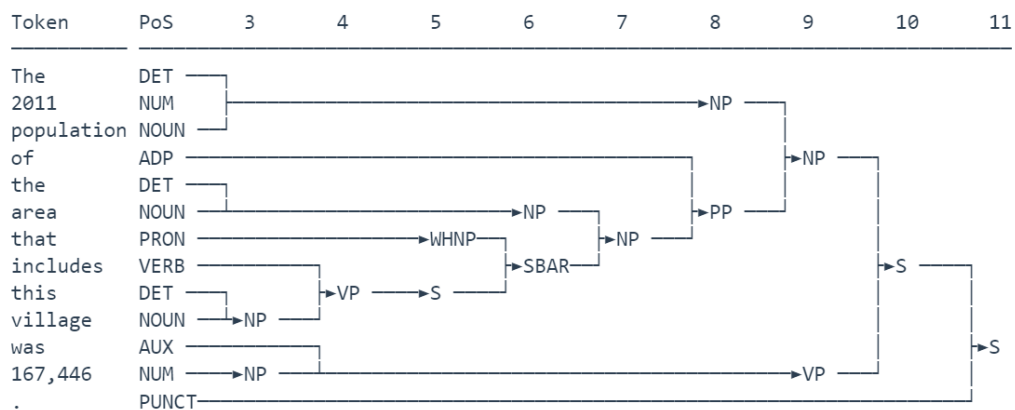
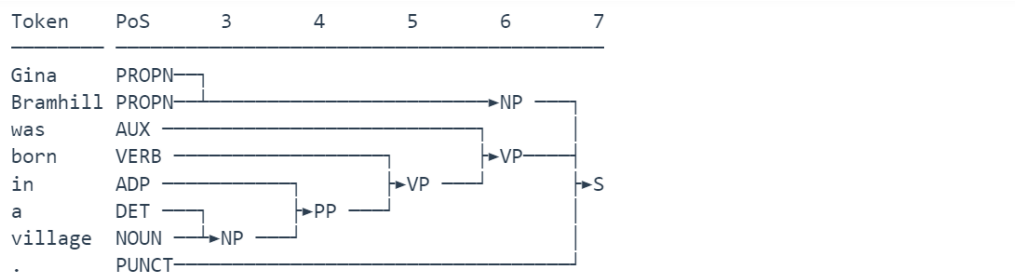
```
Step 1: Ask which song recorded by Fergie was produced by Polow
da Don. answer_1 = Question("Which song recorded by Fergie was
produced by Polow da Don?") # Step 2: Verify if the identified
song was followed by Life Goes On. fact_1 = Verify(f"{answer_1}
was followed by Life Goes On.") # Step 3: Verify if the
identified song was M.I.L.F.$. fact_2 = Verify(f"{answer_1} was M
.I.L.F.$.") # Step 4: Combine the results label = Predict(
fact_1 and fact_2)"
```

```

},
{
 "role": "user",
 "content": "The suggested analysis is: Using a top-down approach,
the sentence is primarily divided into a noun phrase(NP)"The song
recorded by Fergie that was produced by Polow da Don and was
followed by Life Goes On" and a verb phrase(VP)"was M.I.L.F. $"
Noun Phrase (NP) : "The song" serves as the main subject of the
sentence. The NP can be further decompose into 3 clauses.
Relative Clause 1: Modified by "recorded by Fergie," where "by
Fergie" acts as a prepositional phrase (PP) pointing to the artist
involved in the recording. Nested Clause: Within this first
clause, there's another clause "that was produced by Polow da Don
." Here, "that" refers back to "the song," "was produced" forms
the verb phrase (VP), and "by Polow da Don" is another
prepositional phrase describing the producer. Relative Clause 2:
"and was followed by 'Life Goes On'" connects with a conjunction "
and." "Was followed" is the verb, and "by 'Life Goes On'" is the
prepositional phrase indicating what followed the song. Verb
Phrase (VP): "M.I.L.F. $" is the complement of "was," completing
the predicate by specifying the song\u2019s title. By decomposing
the original sentence, we can derive two simplified sub-sentences
:1. M.I.L.F. $ was recorded by Fergie 2. M.I.L.F. $ was produced
by Polow da Don 3. M.I.L.F. $ was followed by 'Life Goes On'
Based on this information, we can transform each fact into a
Verify() function. The first Verify() function checks if the "M.I.
L.F.$ was recorded by Fergie", the second Verify() function checks
if "M.I.L.F.$ was produced by Polow da Don", and the third Verify
() function checks if "M.I.L.F.$ was followed by Life Goes On".
Finally, the Predict() function is used to predict the veracity
label of the overall claim. The logical operator 'and' is used to
combine the results of the three Verify() functions. If all Verify
() functions return true, then the Predict() function will also
return true, meaning the overall claim is true.
the correct answer is: fact_1 = Verify("M.I.L.F.$ was
recorded by Fergie.") fact_2 = Verify("M.I.L.F.$ was produced
by Polow da Don.") fact_3 = Verify("M.I.L.F.$ was followed
by Life Goes On.") label = Predict(fact_1 and fact_2 and
fact_3) . Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},
{
 "role": "user",
 "content": "The claim is that Gina Bramhill was born in a village
. The 2011 population of the area that includes this village was
167,446.. The constituency parsing tree is:

```





```
},
{
```

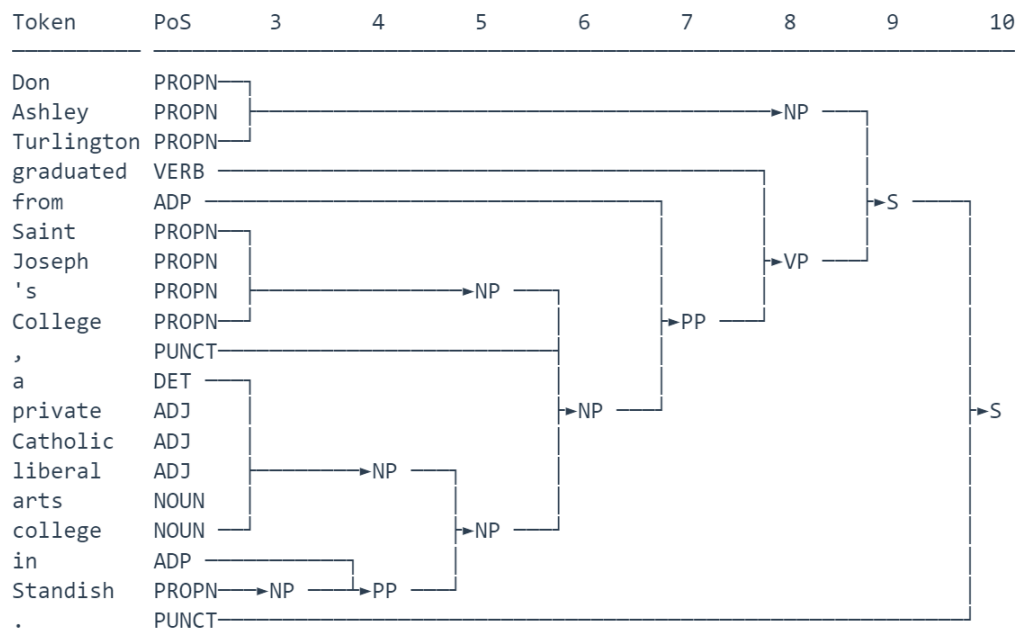
```
 "role": "assistant",
 "content": "Analysis: To verify the claim \"Gina Bramhill was born
in a village. The 2011 population of the area that includes this
village was 167,446,\" we need to break it down into simpler
components based on the constituency tree. The tree shows that the
claim consists of two main parts: 1. Gina Bramhill was born in a
village. 2. The 2011 population of the area that includes this
village was 167,446. From this analysis, we can derive simpler
sub-sentences: 1. Gina Bramhill was born in a village. 2. The
2011 population of the area that includes the village where Gina
Bramhill was born was 167,446. We can transform each fact into
a Verify() function. The first Verify() function checks if \"Gina
Bramhill was born in a village.\" The second Verify() function
checks if \"The 2011 population of the area that includes the
village where Gina Bramhill was born was 167,446.\" Finally, the
Predict() function is used to predict the veracity label of the
overall claim. The logical operator 'and' is used to combine the
results of the two Verify() functions. If both Verify() functions
return true, then the Predict() function will also return true,
meaning the overall claim is true. Program Code: # Step 1:
Verify if Gina Bramhill was born in a village. fact_1 = Verify(\"
Gina Bramhill was born in a village.\") # Step 2: Verify if the
2011 population of the area that includes the village where Gina
Bramhill was born was 167,446. fact_2 = Verify(\"The 2011
population of the area that includes the village where Gina
Bramhill was born was 167,446.\") # Step 3: Combine the results
label = Predict(fact_1 and fact_2)\"
```

```
},
{
```

```

"role": "user",
"content": "The suggested analysis is: The claim involves two
facts about Gina Bramhill and the village she was born in. The
first fact is that "Gina Bramhill was born in a village" and the
second fact is that "The 2011 population of the area that includes
this village was 167,446". The constituency parsing tree helps us
break down the sentence into its constituent parts. The tree
shows that the sentence has a main branch that represents the
first fact: "Gina Bramhill was born in a village". The village is
further described in the second fact: "The 2011 population of the
area that includes this village was 167,446". From the parse tree,
we can identify the subject of the first fact: "Gina Bramhill" (
NP), the action itself: "was born" (VERB), and the location of the
action: "in a village" (PP). We can pose the first part of the
claim as a question using the Question() function: "Which village
was Gina Bramhill born in?". This question aims to identify the
specific village where Gina Bramhill was born. The answer to this
question is stored in the variable answer_1. Then, we use this
answer to verify the second fact that "The 2011 population of the
area that includes {answer_1} was 167,446". This is done using the
Verify() function and stored in the variable fact_1. Finally, we
use the Predict() function to predict the veracity label of the
overall claim. The Predict() function uses the result of the
Verify() function (fact_1) as an input. If the Verify() function
returns true, then the Predict() function will also return true,
meaning the overall claim is true., and the correct answer is:
 answer_1 = Question("Which village was Gina Bramhill born in?")
 fact_1 = Verify(f"The 2011 population of the area that
includes {answer_1} was 167,446.") label = Predict(fact_1).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
"role": "user",
"content": "The claim is that Don Ashley Turlington graduated
from Saint Joseph's College, a private Catholic liberal arts
college in Standish. The constituency parsing tree is:

```



```
},
{
```

```
 "role": "assistant",
 "content": "Analysis: To verify the claim \"Don Ashley Turlington graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish,\" we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of several key elements: 1. The subject: \"Don Ashley Turlington\". 2. The action: \"graduated\". 3. The location: \"from Saint Joseph's College\". 4. Additional information about the college: \"a private Catholic liberal arts college in Standish\".
```

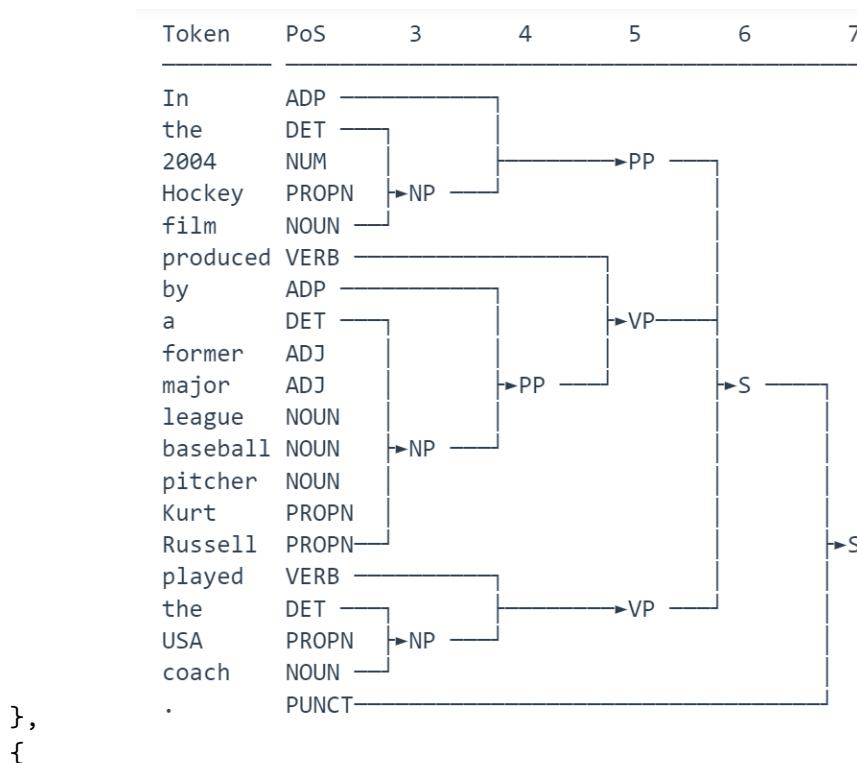
```
From this analysis, we can derive simpler sub-sentences: 1. Don Ashley Turlington graduated from Saint Joseph's College. 2. Saint Joseph's College is a private Catholic liberal arts college in Standish. We can pose the first part of the claim as a question using the Question() function: \"Which college did Don Ashley Turlington graduate from?\". This question aims to identify the specific college from which Don Ashley Turlington graduated. The answer to this question is stored in the variable `answer_1`. Then, we use this answer to verify the second fact that \"{answer_1} is a private Catholic liberal arts college in Standish\". This is done using the Verify() function and stored in the variable `fact_1`. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (`fact_1`) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.
```

```

Program Code: # Step 1: Ask which college Don Ashley Turlington
graduated from. answer_1 = Question("Which college did Don Ashley
Turlington graduate from?") # Step 2: Verify if the identified
college is a private Catholic liberal arts college in Standish.
fact_1 = Verify(f"{answer_1} is a private Catholic liberal arts
college in Standish.") # Step 3: Combine the results label =
Predict(fact_1)"
},
{
 "role": "user",
 "content": "The suggested analysis is: We use a top-down approach
to examine the structure as outlined by the tree. The sentence is
organized around a central noun phrase(NP)"Don Ashley Turlington"
and a verb phrase (VP) "graduated from Saint Joseph's College, a
private Catholic liberal arts college in Standish". Noun Phrase (
NP) "Don Ashley Turlington":This NP serves as the subject of the
sentence, identifying the individual involved. Verb Phrase (VP)
"graduated from Saint Joseph's College, a private Catholic liberal
arts college in Standish": The action "graduated" is the main
verb. It links the subject to the institution from which they
graduated, which is Prepositional Phrase (PP) "from Saint Joseph's
College, a private Catholic liberal arts college in Standish".
Appositive NP "a private Catholic liberal arts college in Standish
" provides additional descriptive information about Saint Joseph\
u2019s College. From the full sentence structure, simplified sub-
sentences can be derived: 1. Don Ashley Turlington graduated from
Saint Joseph's College. 2.Saint Joseph's College is a private
Catholic liberal arts college in Standish. Based on this
information, we can create two Verify() functions. The first
Verify() function checks if "Saint Joseph's College is a private
Catholic liberal arts college in Standish". This fact is derived
from the appositive phrase that describes the college. The second
Verify() function checks if "Don Ashley Turlington graduated from
Saint Joseph's College". This fact is directly stated in the main
clause of the sentence. Finally, the Predict() function is used to
predict the veracity label of the overall claim. The logical
operator 'and' is used to combine the results of the two Verify()
functions. If both Verify() functions return true, then the
Predict() function will also return true, meaning the overall
claim is true., and the correct answer is: fact_1 = Verify("
Saint Joseph's College is a private Catholic liberal arts college
in Standish.") fact_2 = Verify(f"Don Ashley Turlington
graduated from Saint Joseph's College.") label = Predict(
fact_1 and fact_2). Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},
{
 "role": "user",

```

"content": "The claim is that In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach.. The constituency parsing tree is:



"role": "assistant",  
 "content": "Analysis: To verify the claim "In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach," we need to break it down into simpler components based on the constituency tree. The tree shows that the claim consists of several key elements: 1. The context : "In the 2004 Hockey film produced by a former major league baseball pitcher". 2. The action: "Kurt Russell played the USA coach".

From this analysis, we can derive simpler sub-sentences: 1. There is a 2004 Hockey film produced by a former major league baseball pitcher. 2. Kurt Russell played the USA coach in that film.

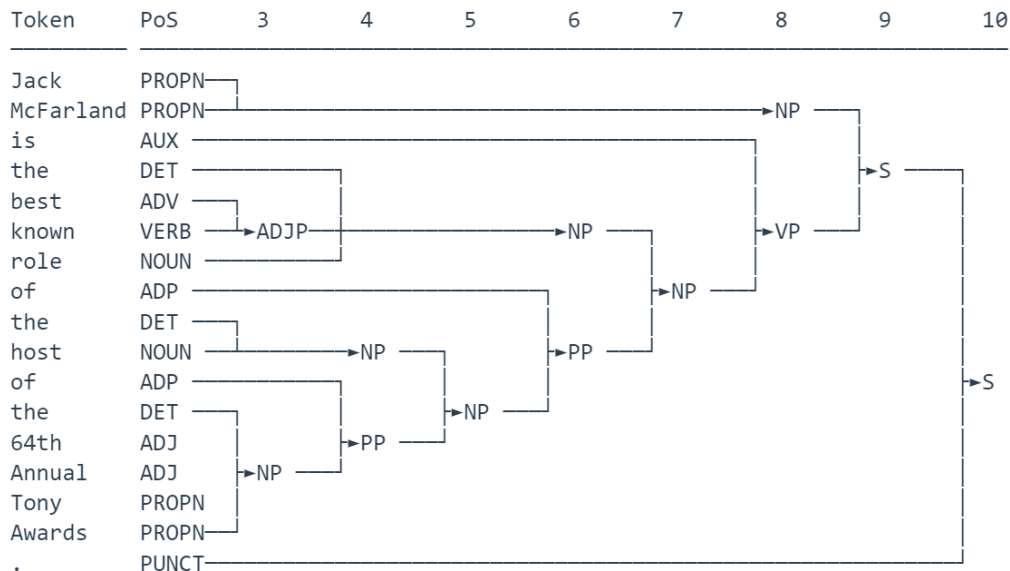
We can pose the first part of the claim as a question using the Question() function: "Which 2004 Hockey film was produced by a former major league baseball pitcher?". This question aims to identify the specific film. The answer to this question is stored in the variable `answer\_1`. Then, we use this answer to verify the second fact that "Kurt Russell played the USA coach in {answer\_1}". This is done using the Verify() function and stored in the variable `fact\_1`. Finally, we use the Predict() function to predict the veracity label of the overall claim. The Predict() function uses the result of the Verify() function (`fact\_1`) as an input. If the Verify() function returns true, then the Predict() function will also return true, meaning the overall claim is true.

```

Program Code: # Step 1: Ask which 2004 Hockey film was produced
by a former major league baseball pitcher. answer_1 = Question("
Which 2004 Hockey film was produced by a former major league
baseball pitcher?") # Step 2: Verify if Kurt Russell played the
USA coach in the identified film. fact_1 = Verify(f"Kurt Russell
played the USA coach in {answer_1}.") # Step 3: Combine the
results label = Predict(fact_1)"
},
{
 "role": "user",
 "content": "The suggested analysis is: The sentence can be
divided into three parts: Prepositional Phrase (PP) 'In the 2004
Hockey film', Verb Phrase (VP) 'produced by a former major league
baseball pitcher', and main Verb (VP) 'played the USA coach'.
Prepositional Phrase (PP): The sentence starts with a
prepositional phrase "In," which sets the temporal and thematic
context. Within this phrase, "the 2004 Hockey film" acts as a noun
phrase (NP) that specifies which film is being discussed. Verb
Phrase (VP): The action "produced" links the film to its producer.
This verb phrase includes a complex prepositional phrase "by a
former major league baseball pitcher," providing detailed
background about the producer. Main Verb (VP): The main clause of
the sentence is "Kurt Russell played the USA coach." Here, "
played" is the main verb indicating the action performed by Kurt
Russell. The object "the USA coach" is a noun phrase describing
the role that Kurt Russell assumed in the film. From the full
sentence structure, simplified sub-sentences can be derived: 1. In
the 2004 Hockey film, Kurt Russell played the USA coach. 2. The
film was produced by Kurt Russell, a former major league baseball
pitcher. To extract the information needed: 1. We use the
Question() function to ask about the specific film: "Which 2004
Hockey film was produced a former major league baseball pitcher?".
This question aims to identify the film produced by the former
pitcher. The answer to this question is stored in the variable
answer_1. 2. Next, we use this answer to verify whether Kurt
Russell played the USA coach in this specific film, forming the
Verify() function: "Kurt Russell played the USA coach in the film
{answer_1}." This checks the accuracy of Kurt Russell's role
in the identified film. Finally, we use the Predict() function to
predict the veracity label of the overall claim based on the
result of the Verify() function. If the Verify() function returns
true, then the Predict() function will also return true, meaning
the overall claim is true., and the correct answer is:
answer_1 = Question("Which 2004 Hockey film was produced a former
major league baseball pitcher?") fact_1 = Verify("Kurt
Russell played the USA coach in the film {answer_1}.") label
= Predict(fact_1). Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
},

```

```
{
 "role": "user",
 "content": "The claim is that Jack McFarland is the best known
role of the host of the 64th Annual Tony Awards. The constituency
parsing tree is:
```



```
},
{
 "role": "assistant",
 "content": "Analysis: To verify the claim \"Jack McFarland is the
best known role of the host of the 64th Annual Tony Awards,\" we
need to break it down into simpler components based on the
constituency tree. The tree shows that the claim consists of
several key elements: 1. The subject: \"Jack McFarland\". 2. The
predicate: \"is the best known role\". 3. The object: \"of the host
of the 64th Annual Tony Awards\".
From this analysis, we can derive simpler sub-sentences: 1.
Identify the host of the 64th Annual Tony Awards. 2. Verify if
Jack McFarland is the best known role of that host.
We can pose the first part of the claim as a question using the
Question() function: \"Who is the host of the 64th Annual Tony
Awards?\". This question aims to identify the specific person who
hosted the 64th Annual Tony Awards. The answer to this question is
stored in the variable `answer_1`. Then, we use this answer to
verify the second fact that \"Jack McFarland is the best known role
of {answer_1}\". This is done using the Verify() function and
stored in the variable `fact_1`. Finally, we use the Predict()
function to predict the veracity label of the overall claim. The
Predict() function uses the result of the Verify() function (`
fact_1`) as an input. If the Verify() function returns true, then
the Predict() function will also return true, meaning the overall
claim is true.
```

```

Program Code: # Step 1: Ask who is the host of the 64th Annual
Tony Awards. answer_1 = Question("Who is the host of the 64th
Annual Tony Awards?") # Step 2: Verify if Jack McFarland is the
best known role of the identified host. fact_1 = Verify(f"Jack
McFarland is the best known role of {answer_1}.") # Step 3:
Combine the results label = Predict(fact_1)"
},
{
 "role": "user",
 "content": "The suggested analysis is: We'll use a top-down
approach to break down the structure based on its components. The
sentence is organized into two main components: a noun phrase (NP)
"Jack McFarland" identifying the subject of the sentence, and a
verb phrase (VP) "is the best known role of the host of the 64th
Annual Tony Awards.", provides additional information about the
subject. Noun Phrase (NP): This NP serves as the main subject of
the sentence. Verb Phrase (VP): The linking verb "is" connects
the subject "Jack McFarland" to the descriptive clause that
follows. The remaining clause could be further divided into NP
"the best known role" and PP "of the host of the 64th Annual Tony
Awards".

From the full sentence structure, simplified sub-sentences can be
derived: 1. Jack McFarland served as the host of the 64th Annual
Tony Awards. 2. Jack McFarland is best known for his role.

We can pose the first part of the claim as a question using the
Question() function: "Who are the hosts of the 64th Annual Tony
Awards?" This question aims to identify the specific host of the
event. The answer to this question is stored in the variable
answer_1. Then, we use this answer to verify the fact that "Jack
McFarland is the best known role of {answer_1}." This is done
using the Verify() function and stored in the variable fact_1.
Finally, we use the Predict() function to predict the veracity
label of the overall claim. The Predict() function uses the result
of the Verify() function (fact_1) as an input. If the Verify()
function returns true, then the Predict() function will also
return true, meaning the overall claim is true.

the correct answer is: answer_1 = Question("Who are the hosts
of the 64th Annual Tony Awards?") fact_1 = Verify(f"Jack
McFarland is the best known role of {answer_1}.") label =
Predict(fact_1). Carefully examine the differences between your
answers and the standard ones. Learn from the correct decomposing
approaches and formats, and apply these insights to your next
claim."
}
]

```

## A.6 Tree-of-Thought with No Supervision(ToT-NS)



As a professional linguistic expert, your final goal is to decompose a complex claim into a program that describes the reasoning steps required to verify the claim. Each function in the program can be seen as a step, and your task is to return one and only one step based on the original claim and the provided pre-decomposed steps.

Note that the generated new step needs to maintain semantic and variable continuity with the preceding steps and avoid semantic repetition. You can call three functions in the program:

1. Question(): answer a question;
2. Verify(): verify a simple claim;
3. Predict(): predict the veracity label.

If no further step is needed, return "TERMINATE". Several decomposed examples are given as follows.

The claim is: WWE Super Tuesday took place at an arena that currently goes by the name TD Garden.

Previous steps are:

```
def program():
 answer_1 = Question("Which arena the WWE Super Tuesday took place
 ?")
```

Next step:

```
fact_1 = Verify(f"{answer_1} currently goes by the name TD Garden
 .")
```

The claim is: Jack McFarland is the best known role of the host of the 64th Annual Tony Awards.

Previous steps are:

```
def program():
 answer_1 = Question("Who is the host of the 64th Annual Tony
 Awards?")
 fact_1 = Verify(f"Jack McFarland is the best known role of {
 answer_1}.")
```

Next step:

```
label = Predict(fact_1)
```

The claim is: Blackstar is the name of the album released by David Bowie that was recorded in secret.

Previous steps are:

```
def program():
 fact_1 = Verify("David Bowie released an album called Blackstar.")
```

Next step:

```
fact_2 = Verify("David Bowie recorded an album in secret.")
```

The claim is: Maria Esther Andion Bueno, not Jimmy Connors, is the player that is from Brazil.

Previous steps are:

```
def program():
```

Next step:

```
fact_1 = Verify("Maria Esther Andion Bueno is from Brazil.")
```

The claim is: The song recorded by Fergie that was produced by Polow da Don and was followed by Life Goes On was M.I.L.F.\$.

Previous steps are:

```
def program():
 fact_1 = Verify("M.I.L.F.$ was recorded by Fergie that was
produced by Polow da Don.")
 fact_2 = Verify("M.I.L.F.$ was was followed by Life Goes On.")
```

Next step:

```
label = Predict(fact_1 and fact_2)
```

The claim is: Talking Heads, an American rock band that was "one of the most critically acclaimed bands of the 80's" is featured in KSPN's AAA format.

Previous steps are:

```
def program():
```

Next step:

```
fact_1 = Verify("Talking Heads is an American rock band that was '
one of the most critically acclaimed bands of the 80's'.")
```

The claim is: An IndyCar race driver drove a Formula 1 car designed by Peter McCool during the 2007 Formula One season.

Previous steps are:

```
def program():
```

Next step:

```
answer_1 = Question("Which Formula 1 car was designed by Peter
McCool during the 2007 Formula One season?")
```

The claim is: Don Ashley Turlington graduated from Saint Joseph's College, a private Catholic liberal arts college in Standish.

Previous steps are:

```
def program():
```

```
fact_1 = Verify("Saint Joseph's College is a private Catholic
liberal arts college is located in Standish.")
```

Next step:

```
fact_2 = Verify(f"Don Ashley Turlington graduated from Saint
Joseph's College.")
```

The claim is: Gael and Fitness are not published in the same country.

Previous steps are:

```
def program():
```

```
answer_1 = Question("Which country was Gael published in?")
answer_2 = Question("Which country was Fitness published in?")
fact_1 = Verify(f"{answer_1} and {answer_2} are not the same
country.")
```

Next step:

```
label = Predict(fact_1)
```

The claim is: Gina Bramhill was born in a village. The 2011 population of the area that includes this village was 167,446.

Previous steps are:

```
def program():
```

```
answer_1 = Question("Which village was Gina Bramhill born in?")
```

Next step:

```
fact_1 = Verify(f"The 2011 population of the area that includes {
answer_1} was 167,446.")
```

The claim is: In the 2004 Hockey film produced by a former major league baseball pitcher Kurt Russell played the USA coach.

Previous steps are:

```
def program():
```

```
 answer_1 = Question("Which 2004 Hockey film was produced a former
major league baseball pitcher?")
```

Next step:

```
 fact_1 = Verify("Kurt Russell played the USA coach in the film {
answer_1}.")
```

The claim is: Along with the New York Islanders and the New York Rangers, the New Jersey Devils NFL franchise is popular in the New York metropolitan area.

Previous steps are:

```
def program():
```

```
 fact_1 = Verify("The New York Islanders and the New York Rangers
are popular in the New York metropolitan area.")
 fact_2 = Verify("The New Jersey Devils NFL franchise is popular in
the New York metropolitan area.")
```

Next step:

```
 label = Predict(fact_1 and fact_2)
```

The claim is: Thomas Loren Friedman has won more Pulitzer Prizes than Colson Whitehead.

Previous steps are:

```
def program():
```

```
 answer_1 = Question("How many Pulitzer Prizes has Thomas Loren
Friedman won?")
 answer_2 = Question("How many Pulitzer Prizes has Colson Whitehead
won?")
```

Next step:

```
 fact_1 = Verify(f"{answer_1} is more than {answer_2}.")
```

The claim is: The model of car Trevor Bayne drives was introduced for model year 2006. The Rookie of The Year in the 1997 CART season drives it in the NASCAR Sprint Cup.

Previous steps are:

```
def program():
```

```
 answer_1 = Question("Which model of car is driven by Trevor Bayne
?")
```

Next step:

```
 fact_1 = Verify(f"{answer_1} was introduced for model year 2006.")
```

The claim is: Barton Mine was halted by a natural disaster not Camlaren Mine.

Previous steps are:

```
def program():
 fact_1 = Verify("Barton Mine was halted by a natural disaster.")
 fact_2 = Verify("Camlaren Mine was not halted by a natural
disaster.")
```

```
Next step:
 label = Predict(fact_1 and fact_2)
```

The claim is: Howard University Hospital and Providence Hospital are both located in Washington, D.C.

Previous steps are:

```
def program():
 fact_1 = Verify("Howard University Hospital is located in
Washington, D.C.")
```

```
Next step:
 fact_2 = Verify("Providence Hospital is located in Washington, D.C
.")
```

The claim is: Vladimir Igorevich Arnold died after Georg Cantor.

Previous steps are:

```
def program():
 answer_1 = Question("When did Vladimir Igorevich Arnold die?")
 answer_2 = Question("When did Georg Cantor die?")
 fact_1 = Verify(f"{answer_1} is after {answer_2}.")
```

```
Next step:
 label = Predict(fact_1)
```

The claim is: Eatza Pizza and Your Pie were not founded in the same state.

Previous steps are:

```
def program():
 answer_1 = Question("Which state was Eatza Pizza founded in?")
```

```
Next step:
 answer_2 = Question("Which state was Your Pie founded in?")
```

The claim is: Gregg Rolie and Rob Tyner, are not a keyboardist.

Previous steps are:

```
def program():
 fact_1 = Verify("Gregg Rolie is not a keyboardist.")
 fact_2 = Verify("Rob Tyner is not a keyboardist.")
```

```
Next step:
 label = Predict(fact_1 and fact_2)
```

The claim is: John O'Hara and Rabindranath Tagore are not the same nationality.

Previous steps are:

```
def program():
 answer_1 = Question("What is the nationality of John O'Hara?")
 answer_2 = Question("What is the nationality of Rabindranath
Tagore?")
 fact_1 = Verify(f"{answer_1} and {answer_2} are not the same
nationality.")
```

```

 label = Predict(fact_1)
Next step:
 TERMINATE

```

\*\*\* Important Rules \*\*\*

- You should never repeat the original claim or previous steps
- You must follow the requirement of ONLY GENERATING ONE STEP.
- You should use variables in previous steps if needed to ensure semantic continuity.
- You should return "TERMINATE" within a python code block once no further steps is needed.

\*\*\* Task Overview \*\*\*

The claim is: {claim}.

Previous steps are:

```

def program():
 {prev_step}

```

Next step:

## A.7 Tree-of-Thought with Supervision(ToT-S)

```

[
 {
 "role": "system",
 "content": "As a professional linguistic expert, your final goal is to decompose a complex claim into a program that describes the reasoning steps required to verify the claim. Each function in the program can be seen as a step, and your task is to return one and ONLY step based on the original claim and the provided pre-decomposed steps. Note that the generated new step needs to maintain semantic and variable continuity with the preceding steps and avoid semantic repetition. You can call three functions in the program:1. Question(): answer a question;2. Verify(): verify a simple claim;3. Predict(): predict the veracity label. *** Important Rules *** - You should never repeat the original claim or previous steps - You must follow the requirement of ONLY GENERATING ONE STEP. - You should use variables in previous steps if needed to ensure semantic continuity. - You should return " TERMINATE" within a python code block once no further steps is needed."
 },
 {
 "role": "user",
 "content": "The claim is: Howard University Hospital and Providence Hospital are both located in Washington, D.C.. Previous steps are: def program(): fact_1 = Verify("Howard University Hospital is located in Washington, D.C.") fact_2 = Verify("Providence Hospital is located in Washington, D.C.") label = Predict(fact_1 and fact_2) Next step:"
 },

```

```

{
 "role": "assistant",
 "content": "```python return label ```"
},
{
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Howard University Hospital and
Providence Hospital are both located in Washington, D.C..
Previous steps are: def program(): fact_1 = Verify("Howard
University Hospital is located in Washington, D.C.") Next step:"
},
{
 "role": "assistant",
 "content": "```python fact_2 = Verify("Providence Hospital is
located in Washington, D.C.") ```"
},
{
 "role": "user",
 "content": "The correct answer is: fact_2 = Verify("Providence
Hospital is located in Washington, D.C."). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: An IndyCar race driver drove a Formula
1 car designed by Peter McCool during the 2007 Formula One season
.. Previous steps are: def program(): answer_1 = Question("
Which Formula 1 car was designed by Peter McCool during the 2007
Formula One season?") Next step:"
},
{
 "role": "assistant",
 "content": "```python answer_2 = Question("Did an IndyCar race
driver drive the Formula 1 car {} during the 2007 Formula One
season?".format(answer_1)) ```"
},
{
 "role": "user",

```

```

"content": "The correct answer is: fact_1 = Verify(f"An
IndyCar race driver drove the car {answer_1}."). Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: An IndyCar race driver drove a Formula
1 car designed by Peter McCool during the 2007 Formula One season
.. Previous steps are: def program(): answer_1 = Question("
Which Formula 1 car was designed by Peter McCool during the 2007
Formula One season?") fact_1 = Verify(f"An IndyCar race
driver drove the car {answer_1}."). Next step:"
},
{
 "role": "assistant",
 "content": "```python label = Predict(fact_1) ```"
},
{
 "role": "user",
 "content": "The correct answer is: label = Predict(fact_1).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: WWE Super Tuesday took place at an
arena that currently goes by the name TD Garden.. Previous steps
are: def program(): answer_1 = Question("Which arena the WWE
Super Tuesday took place?") fact_1 = Verify(f"{answer_1}
currently goes by the name TD Garden.") Next step:"
},
{
 "role": "assistant",
 "content": "```python label = Predict(fact_1) ```"
},
{
 "role": "user",
 "content": "The correct answer is: label = Predict(fact_1).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: WWE Super Tuesday took place at an
arena that currently goes by the name TD Garden.. Previous steps
are: def program(): answer_1 = Question("Which arena the WWE
Super Tuesday took place?") Next step:"

```

```

},
{
 "role": "assistant",
 "content": "```python fact_1 = Verify(f"{answer_1} currently goes
by the name TD Garden.") ```"
},
{
 "role": "user",
 "content": "The correct answer is: fact_1 = Verify(f"{answer_1}
} currently goes by the name TD Garden."). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Talking Heads, an American rock band
that was "one of the most critically acclaimed bands of the 80's"
is featured in KSPN's AAA format.. Previous steps are: def
program(): fact_1 = Verify("Talking Heads is an American rock
band that was 'one of the most critically acclaimed bands of the
80's'.") fact_2 = Verify("Talking Heads is featured in KSPN's
AAA format.") label = Predict(fact_1 and fact_2) Next step
:"
},
{
 "role": "assistant",
 "content": "```python return label ```"
},
{
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Talking Heads, an American rock band
that was "one of the most critically acclaimed bands of the 80's"
is featured in KSPN's AAA format.. Previous steps are: def
program(): fact_1 = Verify("Talking Heads is an American rock
band that was 'one of the most critically acclaimed bands of the
80's'.") Next step:"
},
{
 "role": "assistant",
 "content": "```python fact_2 = Verify("Talking Heads is featured
in KSPN's AAA format.") ```"
},
{

```



```

 "role": "user",
 "content": "The correct answer is: fact_2 = Verify("Talking
Heads is featured in KSPN's AAA format."). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
 },
 {
 "role": "user",
 "content": "The claim is: The model of car Trevor Bayne drives
was introduced for model year 2006. The Rookie of The Year in the
1997 CART season drives it in the NASCAR Sprint Cup.. Previous
steps are: def program(): answer_1 = Question("Which model
of car is driven by Trevor Bayne?") fact_1 = Verify(f"{
answer_1} was introduced for model year 2006.") answer_2 =
Question("Who is the Rookie of The Year in the 1997 CART season?")
 fact_2 = Verify(f"{answer_2} drives {answer_1} in the NASCAR
Sprint Cup.") label = predict(fact_1 and fact_2) Next step
:"
 },
 {
 "role": "assistant",
 "content": "```python return label ```"
 },
 {
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
 },
 {
 "role": "user",
 "content": "The claim is: The model of car Trevor Bayne drives
was introduced for model year 2006. The Rookie of The Year in the
1997 CART season drives it in the NASCAR Sprint Cup.. Previous
steps are: def program(): Next step:"
 },
 {
 "role": "assistant",
 "content": "```python answer_1 = Question("Which model of car is
driven by Trevor Bayne?") ```"
 },
 {
 "role": "user",
 "content": "The correct answer is: answer_1 = Question("Which
model of car is driven by Trevor Bayne?"). Carefully examine the
differences between your answers and the standard ones. Learn from
the correct decomposing approaches and formats, and apply these
insights to your next claim."
 },
 },

```

```

{
 "role": "user",
 "content": "The claim is: The song recorded by Fergie that was
produced by Polow da Don and was followed by Life Goes On was M.I.
L.F.$.. Previous steps are: def program(): fact_1 = Verify
("M.I.L.F.$ was recorded by Fergie.") fact_2 = Verify("M.I.L.
F.$ was produced by Polow da Don.") fact_3 = Verify("M.I.L.F.
$ was was followed by Life Goes On.") label = Predict(fact_1
and fact_2 and fact_3) Next step:"
},
{
 "role": "assistant",
 "content": "`python TERMINATE `"
},
{
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: The song recorded by Fergie that was
produced by Polow da Don and was followed by Life Goes On was M.I.
L.F.$.. Previous steps are: def program(): Next step:"
},
{
 "role": "assistant",
 "content": "`python fact_1 = Verify("M.I.L.F.$ was recorded by
Fergie.") `"
},
{
 "role": "user",
 "content": "The correct answer is: fact_1 = Verify("M.I.L.F.$ was
recorded by Fergie."). Carefully examine the differences between
your answers and the standard ones. Learn from the correct
decomposing approaches and formats, and apply these insights to
your next claim."
},
{
 "role": "user",
 "content": "The claim is: Gina Bramhill was born in a village.
The 2011 population of the area that includes this village was
167,446.. Previous steps are: def program(): answer_1 =
Question("Which village was Gina Bramhill born in?") Next step:"
},
{
 "role": "assistant",
 "content": "`python fact_1 = Verify("The 2011 population of the
area that includes {} was 167,446.".format(answer_1)) `"

```

```

},
{
 "role": "user",
 "content": "The correct answer is: fact_1 = Verify(f"The 2011
population of the area that includes {answer_1} was 167,446.").
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Gina Bramhill was born in a village.
The 2011 population of the area that includes this village was
167,446.. Previous steps are: def program(): answer_1 =
Question("Which village was Gina Bramhill born in?") fact_1 =
Verify(f"The 2011 population of the area that includes {answer_1}
was 167,446.") label = Predict(fact_1) Next step:"
},
{
 "role": "assistant",
 "content": "```python TERMINATE ```"
},
{
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Don Ashley Turlington graduated from
Saint Joseph's College, a private Catholic liberal arts college in
Standish.. Previous steps are: def program(): Next step
:"
},
{
 "role": "assistant",
 "content": "```python fact_1 = Verify("Don Ashley Turlington
graduated from Saint Joseph's College.") ```"
},
{
 "role": "user",
 "content": "The correct answer is: fact_1 = Verify("Saint Joseph's
College is a private Catholic liberal arts college in Standish.")
. Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
 "role": "user",

```

```

 "content": "The claim is: Don Ashley Turlington graduated from
Saint Joseph's College, a private Catholic liberal arts college in
Standish.. Previous steps are: def program(): fact_1 =
Verify("Saint Joseph's College is a private Catholic liberal arts
college in Standish.") Next step:"
 },
 {
 "role": "assistant",
 "content": "```python fact_2 = Verify("Don Ashley Turlington
graduated from Saint Joseph's College.") ```"
 },
 {
 "role": "user",
 "content": "The correct answer is: fact_2 = Verify(f"Don
Ashley Turlington graduated from Saint Joseph's College.").
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
 },
 {
 "role": "user",
 "content": "The claim is: In the 2004 Hockey film produced by a
former major league baseball pitcher Kurt Russell played the USA
coach.. Previous steps are: def program(): Next step:"
 },
 {
 "role": "assistant",
 "content": "```python answer_1 = Question("What is the name of
the 2004 Hockey film produced by a former major league baseball
pitcher?") ```"
 },
 {
 "role": "user",
 "content": "The correct answer is: answer_1 = Question("Which 2004
Hockey film was produced a former major league baseball pitcher
?"). Carefully examine the differences between your answers and
the standard ones. Learn from the correct decomposing approaches
and formats, and apply these insights to your next claim."
 },
 {
 "role": "user",
 "content": "The claim is: In the 2004 Hockey film produced by a
former major league baseball pitcher Kurt Russell played the USA
coach.. Previous steps are: def program(): answer_1 =
Question("Which 2004 Hockey film was produced a former major
league baseball pitcher?") fact_1 = Verify("Kurt Russell
played the USA coach in the film {answer_1}.") Next step:"
 },
 {
 "role": "assistant",
 "content": "```python label = Predict(fact_1) ```"
 }

```

```

},
{
 "role": "user",
 "content": "The correct answer is: label = Predict(fact_1).
Carefully examine the differences between your answers and the
standard ones. Learn from the correct decomposing approaches and
formats, and apply these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Jack McFarland is the best known role
of the host of the 64th Annual Tony Awards. Previous steps are:
def program(): answer_1 = Question("Who are the hosts of the
64th Annual Tony Awards?") fact_1 = Verify(f"Jack McFarland
is the best known role of {answer_1}.) label = Predict(fact_1
) Next step:"
},
{
 "role": "assistant",
 "content": "`python TERMINATE `"
},
{
 "role": "user",
 "content": "The correct answer is: TERMINATE. Carefully examine
the differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
},
{
 "role": "user",
 "content": "The claim is: Jack McFarland is the best known role
of the host of the 64th Annual Tony Awards.. Previous steps are:
def program(): Next step:"
},
{
 "role": "assistant",
 "content": "`python answer_1 = Question("Who are the hosts of
the 64th Annual Tony Awards?") `"
},
{
 "role": "user",
 "content": "The correct answer is: answer_1 = Question("Who are
the hosts of the 64th Annual Tony Awards?"). Carefully examine the
differences between your answers and the standard ones. Learn
from the correct decomposing approaches and formats, and apply
these insights to your next claim."
}
]

```