4-2023

# EXTENDING TEXT2ALM WITH XCLINGO

Joel Sare

# **EXTENDING** TEXT2ALM **WITH** XCLINGO

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment
of the Requirements for the Degree

Master of Science in Computer Science

University of Nebraska at Omaha

by

Joel Sare

April 2023

Supervisory Committee:

Dr. Jorge Fandinno

Dr. Yuliya Lierler

Dr. William Melanson

# **EXTENDING** TEXT2ALM **WITH** XCLINGO

Joel Sare, MS

University of Nebraska, 2023

Advisor: Dr. Jorge Fandinno

This research project goal is to achieve explainable automatic text understanding and reasoning for simple English narratives focused on the use of action verbs. TEXT2ALM is a system developed at UNO in the scope of a master thesis by Craig Olson. It takes a narrative in English and processes it into a logic program under answer set semantics. This program is then processed with the state-of-the-art answer set solver CLINGO, which finds an answer set for the program that represents entities and events occurring within the narrative as well as encodes state of affairs at various points of the narrative timeline. In other words, TEXT2ALM can be seen as an information retrieval system that converts unstructured information present explicitly and implicitly in English narratives into structured form supported by the predefined vocabulary of relations within TEXT2ALM. TEXPLAIN, a system designed at UNO within MS project by Adrian Dorsey, extends TEXT2ALM by replacing answer set solver CLINGO with system XCLINGO. XCLINGO is a novel tool in answer set solving that is capable of providing human readable explanations to inferences performed by CLINGO. In comparison to TEXT2ALM, TEXPLAIN produces explanations for extracted information from narratives. We improve the TEXPLAIN system by allowing automation of creating narratives, queries, logic programs, and XCLINGO output. In this work, we target rigorous evaluation of TEXPLAIN capabilities by applying it on the Facebook dataset bAbI. This dataset contains simple narratives which TEXT2ALM targets. These narratives are annotated with questions, answers, and explanations. The quality of the explanations generated by TEXPLAIN are evaluated against bAbI 's annotations. This evaluation will also lead to extensions of TEXPLAIN capabilities and better understanding of methodological use of system XCLINGO.

*Dedicated to my family*

# *Acknowledgements*

I would like to first thank my thesis committee members Dr. Yuliya Lierler, Dr. Jorge Fandinno, and Dr. William Melanson. I met Dr. Lierler two years ago and she was the first person to really help me start writing research papers. Without her, this document would not be the same. Dr. Fandinno's expertise in answer set programming was invaluable for this Thesis. Without him, tExplain would not be as capable as it is today. I could not have done this without my committee. I would also like to thank Brais Muñiz Castro from the University of A Coruña for answering any questions and resolving any bugs I encountered with xclingo. I would also like to thank my fellow University of Nebraska at Omaha researcher Zachary Hansen for his support. Lastly, I want to thank my friends and family that stood by my side and supported me through this process of writing my Thesis. I couldn't have done this without them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In this work, we design a narrative understanding tool named tExplain. This tool takes as an input a narrative in natural English language that contains both sentences describing actions as well as questions, regarding the entities described in the narrative. Consider below as an example:

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 Mary went to the bathroom.
4 John moved to the hallway.
5 Mary travelled to the office.
6 Daniel put down the apple there.
7 Where is the apple?
```

The output of tExplain are the answers to the queries contained in the narrative together with a natural language explanation about the reasoning the system followed to achieve that conclusion. Below it is an output example obtained from tExplain given the above narrative regarding where the location of the apple is:

```
Answer 1
  *
  |__apple is located at bedroom
  |  |__daniel moved to bedroom in sentence 1
  |  |__since daniel obtained the apple in sentence 2

  *
  |__apple is located at bedroom
  |  |__daniel moved to bedroom in sentence 1
  |  |__since daniel dropped the apple in sentence 6
```

This answer states that the apple is located at the bedroom and shows two different reasons how that can be deduced. The first, by noting that Daniel moved to the bedroom and, then, picked the apple. Since Daniel did not move afterwards, the apple is still in the bedroom.

The second explanation focus on the fact that Daniel dropped the apple while he was in the bedroom.

In building tExplain, we leverage different existing tools such as text2alm and xclingo. Both text2alm and xclingo are built on the backbone of Answer Set Programming (Brewka et al., 2011) or ASP — a knowledge representation declarative programming paradigm with roots in deductive databases, logic programming and satisfiability testing. The system text2alm produces an ASP program from natural language narratives whose solutions, called answer sets, can be used to answer several queries about the narrative. xclingo (Aguado et al., 2019) is an ASP based tool that constructs explanations for found answer sets based on the rules occurring in a program. It also allows users to annotate their programs with snippets of text to produce natural language explanations. System tExplain, developed in this work, combines xclingo and text2alm to provide natural language answers and explanations for questions about natural language narratives. In more detail, the contributions of this work are

- Automatize all the steps of tExplain, from the natural language narrative to the natural language answer and explanation. The preliminary work on tExplain relied on several independent tools that need to be run by the user. The user also needed to perform several manual transformations to the data to make it adequate to the next tool in the pipe line. The tool developed in this work automatizes all this process, so it can be run end-to-end.
- The natural language understanding capabilities of the tool are also expanded to deal with bAbI benchmark Tasks 2, 3, and 6. Tasks 2 and 3 involve describing where an entity is located and using two or three supporting explanations, respectively. Task 6 involves answering yes/no questions about the locations of entities throughout the narrative.
- tExplain also is benchmarked on these tasks and achieved 100% accuracy.

# Chapter 2

# Background

This section will serve as an introduction to logic programs and answer set semantics, TEXT2ALM, XCLINGO, and the first iteration of TEXPLAIN.

## 2.1 Logic Programs and the Answer Set Semantics

Answer Set Programming (ASP) is a form of declarative programming used primarily for NP-complete search problems within knowledge-intensive applications (Lifschitz, 2019). ASP is based on the stable model (or answer set) semantics of logic programming. With ASP, problems are reduced to computing stable models, and programs (answer set solvers) that generate these stable models are used to perform the search.

## 2.2 TEXT2ALM

This section will be a brief overview of TEXT2ALM by considering a sample input and resulting output from the system. TEXT2ALM takes narratives in natural language as input. Here is a shortened narrative from section 1 which is referred to as *narrative 1*:

narrative 1

```
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
```

Given this narrative, TEXT2ALM outputs facts about this text in a structured form that machines can interpret. Below is a subset of the TEXT2ALM output that is grouped together for readability by related groups:

```
Group 1:
    instance(e1, move),
    instance(e2, move),
    instance(e3, move)
Group 2:
    event_agent(e1, mary),
    event_agent(e2, john),
    event_agent(e3, mary)
    event_destination(e1, bathroom),
    event_destination(e2, hallway),
    event_destination(e3, office)
Group 3:
    occurs(e1, 0),
    occurs(e2, 1),
    occurs(e3, 2)
Group 4:
    location(mary, bathroom, 1),
    location(mary, bathroom, 2),
    location(mary, office, 3),
    location(john, hallway, 2),
    location(john, hallway, 3)
```

The first group tells us basic information about the three events that happened within the narrative, namely, `e1`, `e2`, and `e3`. For example, the three events in this narrative are instances of action `move`. The TEXT2ALM tool relies on the VERBNET (Levin, 1993; Palmer, 2018) ontology to identify different verbs occurring in given texts with specific actions. Note how different verbs occurring in our sample narrative, i.e., *went*, *moved*, *travel*, are mapped by TEXT2ALM into the same action `move`. The `event_agent` and `event_destination` facts of Group 2 provide us with the details of the participants of events `e1`, `e2`, and `e3`. For example, `event_agent(e1, mary)` and `event_destination(e1, bathroom)` tell us that `mary` is the acting entity of the `move` action `e1` and bathroom is the destination of `Mary` in the scope of `e1`. Facts, for example `occurs(e1,0)`, contain information pertaining to the time point associated with event `e1`. Within the TEXT2ALM framework, each sentence is associated with a time point value, where `0` denotes the time point for the first sentence within the narrative. For every sentence following the first, the time point is incremented by one. The facts within Group 4 tell us the location of various entities within the narrative for time points `1`, `2`, and `3`. Let us look at the first fact within Group 4: `location(mary, bathroom, 1)`.

This fact contains information that Mary is located in the `bathroom` at time point `1`. This information is associated with the completion of the first sentence within the narrative. There are more conclusions than what is shown in these four groups as TEXT2ALM encodes more information for the full answer set given this three-sentence narrative. For example, looking at Group 1, one can assume that the events `e1`, `e2`, and `e3` relate to the idea of "movement".

There is a defining distinction between the facts in Groups 1-3 and Group 4 pertaining to their nature. The facts in Groups 1-3 stem directly from text. TEXT2ALM uses various natural language processing tools to retrieve information and generate these facts directly. However, for Group 4, TEXT2ALM uses knowledge contained in the CoreALM Library. This library contains axioms for multiple actions which give TEXT2ALM commonsense knowledge about behavior. For our input narrative, axioms for the action `move` are used. TEXT2ALM uses facts within Groups 1-3, common sense knowledge, and the law of inertia to obtain information that is implicitly expressed within a narrative to generate the facts occurring in Group 4. This information allows us to use TEXT2ALM to answer questions like: *"where was John at the end of the story"*? The answer to this is the *hallway* because John explicitly moved to the hallway in sentence two and there's no information to show that he moved somewhere else after sentence three. We can obtain this answer from TEXT2ALM by looking at fact `location(john, hallway, 3)` in Group 4 of its output.

Figure 2.1 shows the building blocks for TEXT2ALM. The Text2LP block is responsible for parsing natural language text, translating information carried by the text into logic program form and expanding it with background knowledge also in the form of a logic program. Thus the Text2LP block translates a given narrative into a logic program in the language of answer set solver Sparc. The answer set solver Sparc translates this program into the format of answer set solver CLINGO (Gebser et al., 2019) and invokes CLINGO to compute answer sets of a given program. The groups of facts presented within our running example stem from computed answer sets.

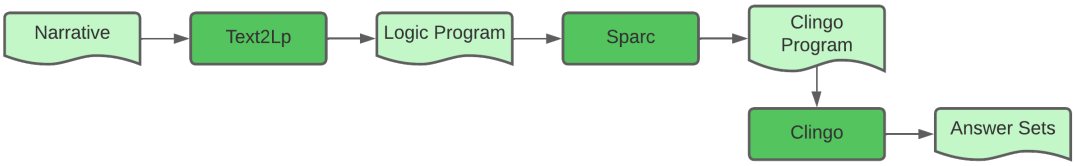


FIGURE 2.1: System TEXT2ALM

## 2.3 XCLINGO

System XCLINGO is a tool in answer set programming that allows the user to create annotations for logic programs so the solved program is shown together along with explanations in natural language. Without these annotations, XCLINGO solves and outputs just like the answer set solver CLINGO (Gebser et al., 2019). We will show XCLINGO annotations with an example. For this example, we will consider an ASP program that is influenced by the first line within *narrative 1*: `Mary went to the bathroom.`

```
instance(e1,move).

event_agent(e1,mary).
event_destination(e1, bathroom).
occurs(e1,0).

location(O,D,T) :- occurs(E,T-1),
                   instance(E,move),
                   event_agent(E,O),
                   event_destination(E,D).
```

The answer set of this program produced by answer set solver CLINGO is below:

```
instance(e1,move)
event_agent(e1,mary)
event_destination(e1,bathroom)
occurs(e1,0)
location(mary,bathroom,1)
```

To transform this output into natural language, we are able to add XCLINGO annotations. For this example, we will focus only on the predicate `location/3`:

```
%!trace {"% is located at % at time %",O,L,T} location (O,L,T).
%!show_trace location(O,L,T).
```

The `%!trace` annotation creates a label for the atom `location/3`. Whenever an atom that has a label is shown in the answer set of a program, an explanation is triggered. In our running example, a label has the form of the tuple

```
{"% is located at % at time %",O,L,T},
```

the first element within this tuple contains a string that serves as the output phrase. The various `%` symbols occurring within this string serve as placeholders for the remaining values in the tuple. For example, the first `%` symbol will get replaced with whatever value varaible `O` has. Atoms of the form `location(O,D,T)` are associated with this `%!trace` annotation within our sample.

Only atoms with a `%!show_trace` annotation are displayed as output. Below is the output by running XCLINGO on our ASP program with the above `%!trace` and `%!show_trace` annotations:

```
Answer: 1
>> location(mary,bathroom,1)        [1]
  *
  |__"mary is located at bathroom at time 1"
```

## 2.4 tExplain 1.0

System TEXT2ALM first produces a logic program that encodes information present in a given narrative and background knowledge required to perform inferences on this narrative; and second applies answer set solver CLINGO to this program to find its answer sets. Figure 4.1 presents an enhanced version of TEXT2ALM titled tExplain that

- introduces annotations for logic programs produced within TEXT2ALM, namely the XCLINGO Trace File which then extend a logic program produced by original TEXT2ALM components and

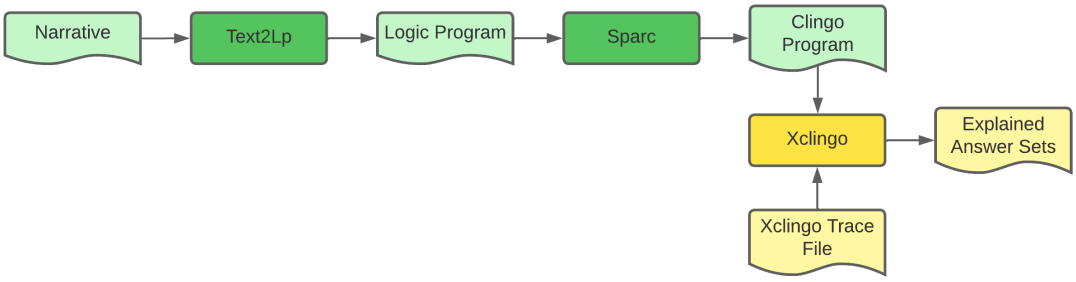


FIGURE 2.2: tEXPLAIN 1.0

- utilizes XCLINGO in place of CLINGO

so that the explanations on relations of interest are provided to the user.

Logic programs produced by the Text2LP block of the TEXT2ALM system are complex and therefore some considerations are required in order to create annotations. In this section, we will show our annotations and some XCLINGO trace rules used in this project. Both trace rules and new predicates are introduced within the XCLINGO Trace File. These predicates play two roles. First, some of these predicates may streamline the form of explanations that we obtain. Second, some of these predicates serve a role of "query-predicates" so that questions about a considered narrative can be formulated by means of the specific instances of these predicates and the help of `%!show_trace` directive of the XCLINGO (see Section 5.3 for an example).

Consider the following rules of XCLINGO Trace File

```
locationT2(Oname,L,T) :- location(Oref,L,T),
                         is_a(Oref,Oname).

%!trace {"% is located at % in sentence %", O, L', T}
    locationT2(O, L, T) : is_a(L, L').
```

The provided trace rule encodes an explanation for location of an entity at a specific sentence. Logic programs produced by the Text2LP block of TEXT2ALM use location argument referents such as `r1` or `r2` instead of names such as "ball" or "mary". These referents and names are linked together through the relation `is_a(_,_)` elsewhere in the program. The first argument in relation `is_a(_,_)` is for the referent and the second is for the English

name. Therefore, we use the auxiliary `is_a(O, O')` together with `O'` instead of `O` in order to instruct the trace to display "ball" and "mary" instead of `r1` and `r2`.

Note how the trace rule is instructed to produce explanations by tracing newly defined predicate `locationT2` in place of `location`. Besides replacing the identifier by its corresponding name, this is essential because predicate `locationT2` is never used within the program produced by text2LP and hence there may not be "nested" explanations due to this predicate. We use `locationT2` and not `location` for explanations because various logic rules that encode background knowledge rely on `location/3` in many ways. Because of this, `location/3`-atoms get used as explanations according to other `location/3`-atoms. This means the XCLINGO system would perform a chain of explanations recursively for every `location/3`-atom. This level of detail is hard to follow and would not be considered natural to human reasoning. Introducing `locationT2/3` allows us to mitigate these recursive explanations.

Section 5.3 provides an example of how `%!show_trace` directive of the XCLINGO can be used for querying an answer set computed by TEXT2ALM driven by specific questions. Here we show an example of the use `%!show_trace`, which selects atoms with predicate symbol `locationT2` as the ones to be be explained:

```
%!show_trace locationT2(O_G,D_G,VAR_0).
```

In TEXPLAIN 1.0 we had an XCLINGO Trace File that allows the user to produce human-readable explanations by transforming logic program data into structured human-readable sentences. Now consider the following group of rules from this XCLINGO Trace File:

```
top_concept(move).
top_concept(obtain).
top_concept(relinquish).
link_r(X,X') :- link(X,X').
link_r(X,X') :- link_r(X,X1),link(X1,X').
event_top_concept(E,C) :- link_r(E,C), top_concept(C).

%!trace {"Since % %ed the % in sentence %",X0', A, B', X2 + 1}
    happened(X1,X2) :
    is_a(X1, X1'),
    event_recipient(X1, X0),
    event_object(X1, B),
    is_a(X0, X0'),
    event_top_concept(X1', A),
    is_a(B, B').
```

The above trace rule is included in order to provide a sub explanation, or justification for location-atoms. This helps when explaining where an object is located at an arbitrary point in a narrative. Therefore, we want to explain the causes. Three `top_concept/`s, 'move', 'obtain', and 'relinquish' are defined. Relation `link_r/2` and `event_top_concept/2` are also defined, with the latter itself being defined in terms of `top_concept` and `link`. The rationale behind these helper rules is that in the answer set program produced by TEXT2ALM before processed by XCLINGO, the immediate events of, for example, an entity picking up an object, are defined using VERBNET terms. In the case of an entity picking up an object, the VERBNET term would be 'get_13_5_1_1'. However, the VERBNET term for grabbing eventually links to the overarching verb 'obtain.' The Text2ALM answer set program contains the notion of `link/2` which we leverage in our `link_r/2` rule in order to propagate the VERBNET term for grabbing up to our `top_concept/1` of 'obtain.' The same applies to the event of an entity moving to a location. The VERBNET term associated with moving gets propagated up to our `top_concept/1` of 'move'. Verbs associated with entities dropping objects are propagated up to our `top_concept/1` of 'relinquish.' This propagation from VERBNET terms to `top_concept/1` allows us to accomplish our goal of generating output in natural language.

The relation `happened/2` encodes an event with a time point. The trace rule includes several `is_a/2` relations so that the explanation can display English words instead of abstractions such as `r1`. Additionally, it utilizes the relations `event_object/2` and

`event_recipient/2` in order to associate the entity that participated in the event `happened/2` with the recipient of the action of the event in `happened/2`. The `event_top_concept/2` relation is defined above the discussed trace rule and is part of XCLINGO Trace File. In total, there are three trace rules in the XCLINGO trace file that help justify the location of entities.

The `%!mute` directive of XCLINGO marks atoms of some form as untraceable. This means the generated explanations will not include/follow them. In the above example, all atoms of the form `dom_location(X,Y,Z)` will not appear in XCLINGO output. In our XCLINGO trace file, there are 86 `%!mute` trace rules as the one below:

```
%!mute dom_location(X,Y,Z).
```

# Chapter 3

# Motivation

The work introducing of tExplain 1.0 was benchmarked against Task 1 (single supporting fact) of bAbI suite (Dorsey, 2021). The first reason to start this project is to extend this work by applying tExplain to Task 2, 3, 6, and 8. It soon became apparent that the human intervention required in tExplain 1.0 did not scale to deal with the increasing number of tasks. As a result, the first task of the project became to fully automatize all tasks that humans need to do with tExplain 1.0. This system was originally split into two parts of text2alm and xclingo with no way to run them together. With the addition of queries and tuples for each narrative processed, the system had many files corresponding to different information. This required the user of the system to know what every component does and how to run the files they generate together. The user will also need to modify files in a specific way to be processed. The new system, tExplain 2.0, introduced in the next section automatizes all these tasks, which include automatically running text2alm, generating queries and tuples, and running xclingo. As a result, the user of the new system tExplain 2.0 can use the system without needing to be an expert on the internal walkabouts of the system.

Once the new system was finished we went back to the original goal of benchmarking tExplain against bAbI Tasks 2, 3, 6, and 8. This requires further enhancements to the xclingo Trace File with new knowledge and trace rules.

# Chapter 4

# tExplain 2.0 (Original Contributions of the Thesis)

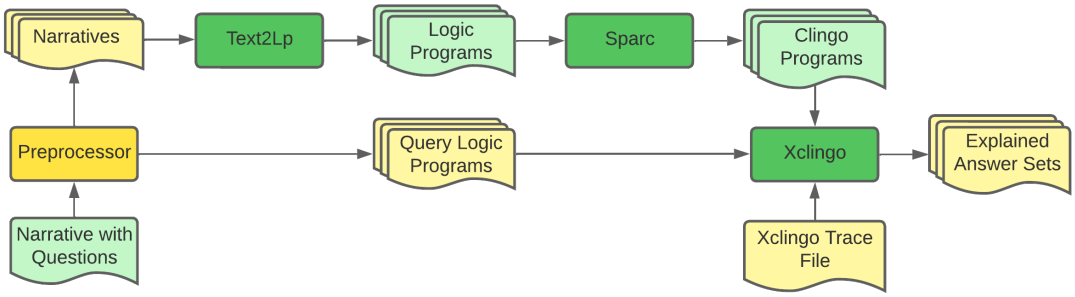


Figure 4.1: tExplain 2.0

In this section, we explore a newer enhanced version of tExplain dubbed tExplain 2.0. tExplain 2.0 builds on tExplain by introducing automation in the form of queries, narratives, and testing on Facebook dataset bAbI as well as new knowledge and trace rules.

## 4.1 Automation

The original system tExplain did not have any form of automation meaning the components from text2alm and xclingo had to be run manually. tExplain 2.0 now automatically creates narratives from an inputted narrative populated with questions, queries from these questions within the narrative, and processes the resulting logic programs generated from text2alm with xclingo creating explained answer sets automatically.

### 4.1.1 Preprocessor

Given one text file containing a narrative with time-framed questions, the preprocessor will create narratives according to how many questions are within the narrative with time-framed three questions. Consider the following narrative with questions,

Narrative with time-framed questions

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 Where is the apple? bedroom 1 2
4 Mary journeyed to the office.
5 Mary left the milk.
6 Where is the milk? office 4 5
7 Daniel went to the hallway.
8 Mary travelled to the bedroom.
9 Where is the apple? hallway 2 7
```

TEXT2ALM and TEXPLAIN 1.0 expect as input a narrative without questions. Therefore, this narrative with questions had to be processed by hand to produce narrative files that correspond to the expected input of these systems. In TEXPLAIN 2.0 this is done automatically. In particular, for the example above, it would create three narrative text files. The first narrative text file consists of the sentences before the first question in the input text file, i.e.:

Narrative 1

```
Daniel went to the bedroom.
Daniel picked up the apple there.
```

The second narrative text file consists of the first narrative text files contents along with the sentences before the second question in the input text file, i.e.:

Narrative 2

```
Daniel went to the bedroom.
Daniel picked up the apple there.
Mary journeyed to the office.
Mary left the milk.
```

This pattern repeats for the rest of the questions within the narrative. Here are the contents of the third narrative text file:

Narrative 3

```
Daniel went to the bedroom.
Daniel picked up the apple there.
Mary journeyed to the office.
Mary left the milk.
Daniel went to the hallway.
Mary travelled to the bedroom.
```

The preprocessor first reads all the lines in the given input narrative one by one sequentially. If the current line is just a sentence, i.e., not a question, it will remove the line number and

output this to the corresponding narrative text file. If the current line is a question, the preprocessor will record the number of questions it has encountered so far then it will close the currently opened narrative text file. Next, it will check if this is the last line in the narrative. If it is not, the preprocessor will open a new narrative text file and begin making the next narrative. If it is, the narrative-making process is terminated for the given input narrative populated with questions.

### 4.1.2 Queries

Instead of filtering through all information within the logic programs created from TEXT2ALM, queries are employed to show only desired information. Specific queries are created depending on the narrative being processed by TEXPLAIN. The bAbI dataset has different tasks and each task has a different question format. These questions are searched for using regular expressions. The matched groups within these regular expressions are used to create various `%!show_trace` queries. In addition to `%!show_trace` queries, there are also facts of the form `query(N)`. These facts mark the line number where questions appear in the narrative with time-framed questions. `query/1` is used in the `mapping/2` rule that allows us to output the correct line number for explanations:

```
mapping(T2,T2+N+1) :- time_point(T2),
                      Q = #max{Q' : nquery(Q',N'), T2 >= Q'-N'},
                      nquery(Q,N).

nquery(Q,N-1) :- query(Q),
                 N = #count{X: query(X), X <= Q}.
```

`mapping/2` takes a line number (`T2`) and maps this to a new line number depending on how many `query/1` rules are present (`T2+N+1`). The rest of this section will discuss query logic programs produced given a bAbI narrative with time-framed questions.

**Two Supporting facts** Questions within this task have the form `Where is the <entity>?` and the regular expression used is `r"(\d+) Where is the (.*)\?"`
Given this input narrative with time-framed questions:

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 Where is the apple?
```

The resulting query logic program produced is:

```
%!show_trace locationT2(apple,L,2).
query(3).
```

**Three Supporting facts** Questions within this task have the form `Where was the <entity> before the <location>?` and the regular expression used is `r"(\d+)? Where was the (.*) before the (.*)\?"`

Given this input narrative with time-framed questions:

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 Daniel journeyed to the bathroom.
4 Where was the apple before the bathroom?
```

The resulting query logic program produced is:

```
%!show_trace lastChangeLoc(apple, L1, bathroom, T).
query(4).
```

**Yes/ No** Questions within this task have the form `Is <entity> in the <location>?` and the regular expression used is `r"(\d+) Is (.*) in the (.*)\?"`

Given this input narrative with time-framed questions:

```
1 Daniel went to the bedroom.
2 Is Daniel in the bedroom?
```

The resulting query logic program produced is:

```
%!show_trace notInLocation(daniel, bedroom, B, 1).
%!show_trace inLocation(daniel, bedroom, B, 1).
is_aB(bedroom, hallway).
query(2).
```

**Lists/ Sets** Questions within this task have the form `What is <entity> carrying?` and the regular expression used is `r"(\d+) What is (.*) carrying\?"`

Given this input narrative with time-framed questions:

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 What is Daniel carrying?
```

The resulting query logic program produced is:

```
%!show_trace entityCarrying(N, daniel, 2).
%!show_trace entityNotCarrying(N, daniel, 2).
query(3).
```

### 4.1.3 Configuration File

The configuration file contains various information about where to store generated files created by the automation tool. The file has information about where to store logic programs, narratives, queries, and XCLINGO output files. This configuration file expects direct file paths on the users computer. This file is essential for narrative/query making and logic/XCLINGO file processing.

### 4.1.4 Processing Tool

This section describes the tool that is able to handle the automatic creation of narratives, queries, logic programs, and XCLINGO output given a single text narrative populated with time-framed questions.

This tool first reads in the narrative to be processed. Then it is processed into narratives as outlined in section 4.1.1. Then TEXT2ALM is run on the processed narrative files separately, resulting in corresponding logic programs. The queries are then processed as outlined in section 4.1.2. Finally, XCLINGO is run on all logic programs and corresponding queries.

**How to run** To run the TEXPLAIN system, issue a command of the form:

```
python runbAbI.py threeSuppA.txt
```

Where `threeSuppA.txt` is a text file containing a bAbI English narrative. There are five bAbI tasks that tExplain is able to automatically process:

**Task 1:** Single Supporting Facts

**Task 2:** Two Supporting Facts

**Task 3:** Three Supporting Facts

**Task 6:** Yes No

**Task 8:** Lists Sets

tExplain will automatically create queries given a narrative from any of the listed bAbI tasks above. Each bAbI task has unique question format and pattern that generate unique queries by using the correct regular expression outlined in section 4.1.2 to extract the desired information to trace. Narratives are created the same no matter which bAbI task is being processed. After issuing the above command, the processing tool will create all files mentioned earlier and generate outputs and explanations in natural language.

# Chapter 5

# bAbI Testing

## 5.1 bAbI dataset

The bAbI dataset (Weston et al., 2015) is provided by Facebook AI Research and it is designed to be a benchmark for automated text understanding and reasoning. The bAbI dataset consists of several tasks for systems to experiment on ranging from yes/ no questions, basic deduction, and path finding. In these tasks, there are entities of various types like locations, objects, people, etc. and various actions can operate upon these entities. These entities also have internal states such as location, whether they have objects on top of or in them, the mental state of actions, and properties such as size, color, and entity. Figures 5.1 and 5.2 provide snippets from two different tasks of bAbI, namely, Task 2 called "two supporting facts" and Task 3 called "three supporting facts". Note how bAbI narratives are simple stories, where various entities perform actions. These narratives also contain questions prompted at various time points.

Now let us examine bAbI questions and the numeric annotations that come with them. Consider Line 7 in Figure 5.1. In this line, bAbI suggests that the answer to *where the apple is located* is in the *bedroom*. The two numbers appearing after word “bedroom” are the two sentences that support this answer. Consequently, we can read this line as saying that

> Upon the completion of the narrative composed of sentences in lines 1 through 6, *apple* is located in *bedroom* because we have been told that *Daniel went to the bedroom* in sentence one and then *he put down the apple* in sentence six.

## 5.2 Evaluation Observations on Task 2

**bAbI Patterns** Clear patterns emerge within the bAbI narrative-question-answer-explanation tuples. This is due to the nature of the bAbI dataset as it has been created shadowing the

```
1 Daniel went to the bedroom.
2 Daniel picked up the apple there.
3 Mary grabbed the milk there.
4 Mary left the milk.
5 John journeyed to the office.
6 Daniel put down the apple there.
7 Where is the apple?  bedroom 6 1
```

FIGURE 5.1: Example entry from the bAbI task called two supporting facts

```
1 Mary moved to the bathroom.
2 Sandra journeyed to the bedroom.
3 Mary got the football there.
4 John went back to the bedroom.
5 Mary journeyed to the office.
6 John journeyed to the office.
7 John took the milk.
8 Daniel went back to the kitchen.
9 John moved to the bedroom.
10 Daniel went back to the hallway.
11 Daniel took the apple.
12 John left the milk there.
13 John travelled to the kitchen.
14 Sandra went back to the bathroom.
15 Daniel journeyed to the bathroom.
16 John journeyed to the bathroom.
17 Mary journeyed to the bathroom.
18 Sandra went back to the garden.
19 Sandra went to the office.
20 Daniel went to the garden.
21 Sandra went back to the hallway.
22 Daniel journeyed to the office.
23 Mary dropped the football.
24 John moved to the bedroom.
25 Where was the football before the bathroom?  office 23 17 5
```

FIGURE 5.2: Example entry from the bAbI task called three supporting facts

simulation environment and transcribing its history Weston et al. (2015). We illustrate the concept of a pattern on an example of Task 2 consisting of 16 sentences and 5 questions. Figure 5.3 contains this sample bAbI narrative in the left column. We can identify the content of this column with *five* narrative-question-answer-explanation tuples. For instance, the narrative of the first tuple consists of sentences in lines 1-6; the narrative of the second tuple consists of sentences in lines 1-6,8, and 9. Lines 7 and 10 enumerate question, answer

| | |
|---|---|
| 1 Daniel went to the bedroom. | 1 Person_1 <move_to> location_1. |
| 2 Daniel picked up the apple there. | 2 Person_1 <grab> object_1. |
| 3 Mary grabbed the milk there. | 3 Person_2 <grab> object_2. |
| 4 Mary left the milk. | 4 Person_2 <drop> object_2. |
| 5 John journeyed to the office. | 5 Person_3 <move_to> location_2. |
| 6 Daniel put down the apple there. | 6 Person_1 <drop> object1. |
| 7 Where is the apple? bedroom 6 1 | 7 Where is object_1? location_1 6 1 |
| 8 John picked up the milk there. | 8 Person_3 <grab> object_2 |
| 9 Sandra got the football there. | 9 Person_4 <grab> object_3. |
| 10 Where is the apple? bedroom 6 1 | 10 Where is object_1? location_1 6 1 |
| 11 Daniel journeyed to the hallway. | 11 Person_1 <move_to> location_2. |
| 12 John left the milk. | 12 Person_3 <drop> object_2. |
| 13 Where is the milk? office 12 5 | 13 Where is object_2? location_2 12 5 |
| 14 Sandra travelled to the office. | 14 Person_4 <move_to> location_2. |
| 15 Sandra put down the football there. | 15 Person_4 <drop> object_3. |
| 16 Where is the football? office 15 14 | 16 Where is object_3? location_2 15 14 |
| 17 Sandra grabbed the milk there. | 17 Person_4 <grab> object_2. |
| 18 John grabbed the football there. | 18 Person_3 <grab> object_3. |
| 19 Sandra moved to the bathroom. | 19 Person_4 <move_to> location_3. |
| 20 John went to the bedroom. | 20 Person_3 <move_to> location_1. |
| 21 Where is the football? bedroom 18 20 | 21 *Where is object_3? location_1 18 20.* |

FIGURE 5.3: Translating bAbI entry within two supporting facts into pattern format

and explanation of the first and second tuples, respectively. In the sequel, we frequently identify a term *question* with a notion of narrative-question-answer-explanation tuple. In the 20 narratives listed within both qa2_two-supporting-facts_train.txt and qa3_three-supporting-facts_train.txt there were total 100 narrative-question-answer-explanation tuples, 50 coming from each file. The right column translates this narrative and questions into generic pattern form. Note how, for instance, distinct verbs *picked up, grabbed, got* occurring in the left column are captured by the same concept <grab> in the right column; also specific names of people, objects, and locations are identified with ids of the form person_p, object_y, and location_z, respectively.

Table 5.1 captures two patterns of narrative-question-answer-explanation tuples. Consider pattern **P1**. Let us examine the *bAbI example* column of Table 5.1. The three declarative sentences **A-C** stem from the narrative consisting of lines 1-6 in Figure 5.3 (left column). The question **Q** coincides with the one listed in line 7 in Figure 5.3 (left column). These three sentences (out of the total six sentences of the considered narrative) appear in **P1** as each of them refers to a pair of entities related to inferences required to find the answer to

given question **Q**. In other words to establish the location of an entity *apple* we have to analyze its relation to entities *Daniel* and *bedroom*. All other sentences preceding the question in line 7 in Figure 5.3 (left column) do not refer to any of the three entities among *apple*, *Daniel*, and *bedroom*. The *Pattern format* column of Table 5.1 uncovers the generic form of the narrative-question-answer-explanation tuple of the considered example. Now note that the first three questions (narrative-question-answer-explanation tuples) in Figure 5.3 (left column) fall into this pattern **P1**. Question in line 21 in Figure 5.3 (left column) falls into pattern **P2** presented in Table 5.1. Question in line 16 in Figure 5.3 (left column) falls into a variant of pattern **P1** that we denote as **P1′**. Pattern **P1′** is formed from **P1** by switching the order of sentences of the form **A** and **B** presented in Table 5.1.

| | **bAbI example** | **bAbI pattern** |
|---|---|---|
| **P1** | **A** Daniel went to the bedroom.<br>**B** Daniel picked up the apple there.<br>**C** Daniel put down the apple there.<br>**Q** Where is the apple? bedroom **A C** | **A** Person_P <move_to> location_X.<br>**B** Person_P <grab> object_Y.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **A C** |
| **P1′** | **A** Sandra got the football there.<br>**B** Sandra travelled to the office.<br>**C** Sandra put down the football there.<br>**Q** Where is the football? office **B C** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_X.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **B C** |
| **P2** | **A** John grabbed the football there.<br>**B** John went to the bedroom.<br>**Q** Where is the football? bedroom **A B** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_Z.<br>**Q** Where is object_Y? location_Z **A B** |

TABLE 5.1: Task 2 Narrative-Question-Answer-Explanation Patterns

As mentioned earlier, we focused on processing 10 narratives listed first in qa2_two-supporting-facts_train.txt, the file stemming from bAbI. Patterns **P1**, **P1′**, **P2** are the only ones occurring within these narratives.

**tExplain Patterns in Task 2** Patterns found within the bAbI dataset resurface in the same manner when narratives are being processed by tExplain. Figure 5.2 presents the behavior of system tExplain on patterns **P1**, **P1′**, and **P2** from bAbI: these bAbI patterns are repeated once more in the left hand side column. For all patterns, the explanations given by tExplain coincide with the ones suggested by bAbI. For patterns **P1** and **P1′** additional explanations are produced by the tExplain system.

| | **bAbI pattern** | tExplain **pattern** |
|---|---|---|
| **P1** | **A** Person_P <move_to> location_X.<br>**B** Person_P <grab> object_Y.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **A C** | **A** Person_P <move_to> location_X.<br>**B** Person_P <grab> object_Y.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **A C**<br>Additional explanation by tExplain: **A B** |
| **P1′** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_X.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **B C** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_X.<br>**C** Person_P <drop> object_Y.<br>**Q** Where is object_Y? location_X **B C**<br>Additional explanation by tExplain: **A B** |
| **P2** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_Z.<br>**Q** Where is object_Y? location_Z **A B** | **A** Person_P <grab> object_Y.<br>**B** Person_P <move_to> location_Z.<br>**Q** Where is object_Y? location_Z **A B** |

Table 5.2: Comparison between bAbI and tExplain with Task 2 patterns

For Task 2, the original system developed in scope of Adrian's master project produced only explanations dealing with entities picking up objects and moving to locations. As an example, consider pattern **P1** under the tExplain pattern column. The old system would explain that object_Y is in location_X by stating person_P moved to location_X in sentence **A** and picked up object_Y in sentence **B**. The previous system would not be able to explain Task 2 with sentences involving dropping objects, particularly sentence **C**. However, with the addition of `must_possesses/3`:

```
must_possesses(D_G,O_G,TS_G) :-
    event_donor(X_G,D_G),
    event_object(X_G,O_G),
    occurs(X_G,TS_G),
    instance(X_G,transfer),
    entity(O_G),universe(X_G),
    entity(D_G),event(X_G),
    timeStep(TS_G),
    actions(X_G).

must_possesses(X0_G,X1_G,VAR_0) :-
    must_possesses(X0_G,X1_G,I_G),
    not -possesses(X0_G,X1_G,VAR_0),
    dom_possesses(X0_G,X1_G,VAR_0),
    VAR_0=I_G-1,
    entity(X0_G),
    entity(X1_G),
    timeStep(I_G),
    timeStep(VAR_0),
    not takes(X0_G,X1_G,VAR_0).

takes(D_G,O_G,VAR_0) :-
    event_recipient(X_G,D_G),
    event_object(X_G,O_G),
    occurs(X_G,VAR_0),
    timeStep(VAR_0),
    take_instance(X_G),
    entity(O_G),universe(X_G),
    entity(D_G),event(X_G).

take_instance(X_G) :- instance(X_G, obtain).

possesses(D_G,O_G,TS_G) :- must_possesses(D_G,O_G,TS_G).
```

we are able to successfully produce two sets of two supporting facts. This rule works by ensuring that if an entity drops an object at a certain time point, they possess that object at that time point and all previous time points as long as the `takes/3` rule is not satisfied. The `takes/3` rule validates these previous time points by ensuring if the entity encounters a `take_instance` they do not posses the object anymore. The `take_instance/1` rule checks for instances where entities grab objects.

These two sets of two supporting facts consist of one set with explanations dealing with the entity moving to a location (sentence **A**) and picking up an object (sentence **B**) and the

other set consists of the entity moving to a location (sentence **A**) and dropping an object (sentence **C**). Both of these sets of explanations are logical in their conclusions and are both considered to be correct. However, bAbI only shows one "correct" explanation for every question-answer-explanation tuple.

## 5.3 Evaluation Observations on Task 3

Recall a sample narrative of Task 3 presented in Figure 5.2. It concludes with the question

*Where was the football before the bathroom?* (5.1)

where the subphrase *before the bathroom* pins the time point to when the location of the football is of interest to us. All questions of bAbI Task 3 are of this form. Because of this, tracking where entities change location is important when answering and providing explanations for the questions of Task 3. We introduce a new predicate within XCLINGO Trace File, namely, `changeLoc/4` and `lastChangeLoc/4`, that tracks entities and timepoints when these location changes happen. The definition of this predicate follows:

```
changeLoc(Oname, L1name, L2name, T) :-
    is_a(Oref,Oname),
    is_a(L1ref, L1name),
    is_a(L2ref, L2name),
    changeLocAux(Oref, L1ref, L2ref, T).

changeLocAux(Oref, L1ref, L2ref, T) :-
    locationB(Oref, L1ref, T-1),
    locationB(Oref, L2ref, T),
    L1ref != L2ref.

lastChangeLocAux(Oref, L1ref, L2ref, T) :-
    changeLocAux(Oref, L1ref, L2ref, T),
    T = #max{X: changeLocAux(Oref, _, L2ref, X)}.

lastChangeLoc(Oname, L1name, L2name, T) :-
    is_a(Oref,Oname),
    is_a(L1ref, L1name),
    is_a(L2ref, L2name),
    lastChangeLocAux(Oref, L1ref, L2ref, T).
```

We specify the *last* location of an object with `lastChangeLoc/4` as we are interested in the latest case in the narrative where a specific object (`Oref`) has changed locations from the location in question (`L2ref`). With larger narratives, the same object can change many different locations, some repeating other time points earlier in the narrative. With this rule we are able to specify only the last time the object has changed locations from the location in question. This is achieved with `changeLocAux/4` which finds the latest time point of an object changing locations that contains information about `Oref` and `L2ref`.

This predicate allows us to compose a query by means of `%!show_trace` directive of XCLINGO given questions of Task 3. For example, the following `%!show_trace` expression captures our sample question (5.1):

```
%!show_trace lastChangeLoc(football, L1, bathroom, T).
```

**bAbI Patterns** As with Task 2, consistent patterns emerge within the Task 3 dataset. This concept is illustrated in Table 5.3, where two patterns of narrative-question-answer-explanation tuples are captured. **P3** consists of an entity moving to two locations and dropping an object. In fact, Figure 5.2 presents narrative-question-answer-explanation tuple that is used to exemplify pattern **P3**. We used the same methodology as in case of uncovering patterns for Task 2 for the decision on which sentences of the considered narrative become part of the pattern for a given question. A sentence from a narrative appears in the pattern if it refers to a pair of entities related to inferences required to find the answer to given question. Consider the narrative-question-answer-explanation tuple in Table 5.3, in order to establish the location of the *football* prior to it being in the *bathroom* we have to analyze its relation to entities *Mary*, *office*, and *bathroom*. Pattern **P4** represents the case when an entity grabs an object first and then moves to two locations.

From the 10 narratives that we processed in qa3_two-supporting-facts_train.txt patterns **P3** and **P4** are the only ones occurring within these narratives.

TEXPLAIN **Patterns in Task 3** Just as in Task 2 evaluation, patterns observed within the bAbI dataset for Task 3 resurface in the same manner when narratives are being processed by system TEXPLAIN. Table 5.4 presents the behavior of TEXPLAIN on patterns **P3** and

| | bAbI example | Pattern format |
|---|---|---|
| **P3** | **A** Mary moved to the bathroom.<br>**B** Mary got the football there.<br>**C** Mary journeyed to the office.<br>**D** Mary journeyed to the bathroom.<br>**E** Mary dropped the football.<br>**Q** Where was the football before the bathroom?<br>office **E D C** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <grab> object_Z.<br>**C** Person_P <move_to> location_X.<br>**D** Person_P <move_to> location_Y.<br>**E** Person_P <drop> object_Z.<br>**Q** Where was object_Z before location_Y?<br>location_X **E D C** |
| **P4** | **A** Sandra went back to the hallway.<br>**B** Sandra travelled to the kitchen.<br>**C** Sandra took the milk.<br>**D** Sandra left the milk there.<br>**E** Sandra picked up the milk.<br>**F** Sandra travelled to the hallway.<br>**G** Sandra went to the kitchen.<br>**Q** Where was the milk before the kitchen?<br>hallway **E G F** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <move_to> location_Z.<br>**C** Person_P <grab> object_X.<br>**D** Person_P <drop> object_X.<br>**E** Person_P <grab> object_X.<br>**F** Person_P <move_to> location_Y.<br>**G** Person_P <move_to> location_Z.<br>**Q** Where was object_X before location_Z?<br>location_Y **E G F** |

TABLE 5.3: Task 3 Narrative-Question-Answer-Explanation Patterns

**P4** (these bAbI patterns are repeated once more in the left hand side column). For both patterns, the explanations given by tEXPLAIN coincide with the ones suggested by bAbI. For pattern **P3**, additional explanations are produced by the tEXPLAIN system.

Pattern **P3** in Table 5.4 under the tEXPLAIN output column shows an extra explanation produced by tEXPLAIN which consists of Person_P grabbing object_Z in sentence **B**, moving to location_X in sentence **C**, and finally moving to location_Y in sentence **D**. This explanation is logical in its conclusion and would be as correct as the only single "correct" bAbI answer provided. The other extra explanation provided by tEXPLAIN consists of sentences **B**, **C**, **D**, *and* **E**. This second additional explanation would also be logical in its conclusion, but not minimal one. The work is limited here in addressing this issue as, unfortunately, XCLINGO does not allow to compute minmal explanations.

## 5.4 Evaluation Observations on Task 6

Task 6 involves answering yes/ no questions. All of these questions deal with identifying the location of a specific entity:

*Is Sandra in the hallway?* (5.2)

As there are only two possibilities of the entity being in the location or the entity not being in the location. We introduce two new predicates within XCLINGO Trace File, namely,

| | bAbI example | tExplain output |
|---|---|---|
| **P3** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <grab> object_Z.<br>**C** Person_P <move_to> location_X.<br>**D** Person_P <move_to> location_Y.<br>**E** Person_P <drop> object_Z.<br>**Q** Where was object_Z before location_Y?<br>location_X **E D C** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <grab> object_Z.<br>**C** Person_P <move_to> location_X.<br>**D** Person_P <move_to> location_Y.<br>**E** Person_P <drop> object_Z.<br>**Q** Where was object_Z before location_Y?<br>location_X **E D C**<br>Also produced by tExplain: **[B C D]**, **[B C D E]** |
| **P4** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <move_to> location_Z.<br>**C** Person_P <grab> object_X.<br>**D** Person_P <drop> object_X.<br>**E** Person_P <grab> object_X.<br>**F** Person_P <move_to> location_Y.<br>**G** Person_P <move_to> location_Z.<br>**Q** Where was object_X before location_Z?<br>location_Y **E G F** | **A** Person_P <move_to> location_Y.<br>**B** Person_P <move_to> location_Z.<br>**C** Person_P <grab> object_X.<br>**D** Person_P <drop> object_X.<br>**E** Person_P <grab> object_X.<br>**F** Person_P <move_to> location_Y.<br>**G** Person_P <move_to> location_Z.<br>**Q** Where was object_X before location_Z?<br>location_Y **E G F** |

Table 5.4: Comparison between bAbI and tExplain with Task 3 patterns

`notInLocation/2` and `inLocation/2`, that tracks entities and locations. The definition of these predicates follows:

```
notInLocation(Oname,Lname,T2,T) :-
    is_a(Oref,Oname),
    is_aB(Lref,Lname),
    location(Oref,Lref',T),
    Lref!=Lref',
    not is_a(Lref',Lname),
    mapping(T,T2).

inLocation(Oname,Lname,T2,T) :-
    is_a(Oref,Oname),
    is_a(Lref,Lname),
    location(Oref,Lref,T),
    mapping(T,T2).
```

In section 5.2, we discussed the idea of narrative-question-answer-explanation tuples. This structure isolates the whole narrative structure into separate smaller narratives. Because of this, some locations were asked about that smaller narratives had no knowledge of. For example, question 5.2 asks if *Sandra* is located in *hallway*. But it may be the case that the location *hallway* was never mentioned in the narrative-question-answer-explanation tuple. For this reason, we create an `is_aB/2` relation with every location asked within a question. In the below example, we query the location of Sandra in the hallway and create a new relation `is_aB/2` for the hallway location:

```
%!show_trace notInLocation(sandra, hallway, 2).
%!show_trace inLocation(sandra, hallway, 2).
is_aB(hallway, hallway).
```

The `is_aB/2` relation provides information of the location *hallway* even if it wasn't mentioned earlier in the narrative. From the 10 narratives that we processed in qa6_yes-no-questions_train.txt consisting of 50 narrative-question-answer-explanation tuples, 19 tuples asked about the location of an entity before that location was mentioned earlier in the narrative. We did not see the case where a narrative-question-answer-explanation tuple asked a question about an entity that was not mentioned earlier in the narrative. System tExplain was able to correctly answer and explain every question from the 10 narratives we processed.

## 5.5 Evaluation Observations on Task 8

Task 8 tests the system with questions pertaining to what entities are carrying at various time points. Consider below as an example narrative with time-framed questions containing only the entity `Daniel`:

```
1 Daniel went to the kitchen.
2 Daniel took the football there.
3 Daniel picked up the milk there.
4 What is Daniel carrying? football,milk 2 3
5 Daniel discarded the milk.
6 What is Daniel carrying? football 2 3 5
7 Daniel left the football there.
8 What is Daniel carrying? nothing 2 7 3 5
```

In Task 8 there are three patterns that arise when answering this question. The entity could either be holding one object, more than one object, or nothing. Table 5.5 shows these three bAbI patterns along with tExplain output. To generate answers and explanations for this task, we introduce three new predicates within the xclingo Trace File, `entityCarrying/3`, `entityNotCarrying/3`, and `entityCarryingSomething/2`. The definitions of these predicates are shown below:

| | **bAbI pattern** | tExplain **pattern** |
|---|---|---|
| **P1** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**Q** What is Person_P carrying?<br>object_X, object_Y **B C** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**Q** What is Person_P carrying?<br>object_X, object_Y **B C** |
| **P2** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**D** Person_P <drop> object_Y.<br>**Q** What is Person_P carrying?<br>object_X **B C D** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**D** Person_P <drop> object_Y.<br>**Q** What is Person_P carrying?<br>object_X **B D** |
| **P3** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**D** Person_P <drop> object_Y.<br>**E** Person_P <drop> object_X.<br>**Q** What is Person_P carrying?<br>nothing **B E C D** | **A** Person_P <move_to> location_Z.<br>**B** Person_P <grab> object_X.<br>**C** Person_P <grab> object_Y.<br>**D** Person_P <drop> object_Y.<br>**E** Person_P <drop> object_X.<br>**Q** What is Person_P carrying?<br>nothing **D E** |

Table 5.5: Comparison between bAbI and tExplain with Task 8 patterns

```
entityCarrying(Lname,Ename,T) :-
    held_by(Lref,Eref,T),
    living_entity(Eref),
    timeStep(T),
    is_a(Eref,Ename),
    is_a(Lref,Lname).

entityNotCarrying(Lname,Ename,T) :-
    -held_by(Lref,Eref,T),
    living_entity(Eref),
    timeStep(T),
    is_a(Eref,Ename),
    is_a(Lref,Lname),
    not entityCarryingSomething(Ename,T).

entityCarryingSomething(Ename,T) :- entityCarrying(Lname,Ename,T).
```

In the example narrative with time-framed questions dealing with entity `Daniel`, tExplain successfully answers and explains the question on line 4 that `Daniel` is carrying both a football and milk because he picked up the football in sentence 2 and then in sentence 3 he picked up the milk. This follows pattern **P1** present in Table 5.5. For the questions on lines 6 and 8, and patterns **P2** and **P3**, tExplain gives different explanations than what bAbI considers to be correct.

The question on line 6 shows an example of pattern **P2**. For this question, bAbI explains that entity `Daniel` is carrying one object, a football, because he took the football in sentence 2, picked up the milk in sentence 3, and then put down the milk in sentence 5. tExplain answers this same question by stating that `Daniel` picked up the football in sentence 2 and he dropped the milk in sentence 5.

For the question on line 8, bAbI explains that `Daniel` is carrying nothing by describing that he picked up the football in sentence 2, picked up the milk in sentence 3, dropped the milk in sentence 5, and then dropped the football in sentence 7. tExplain explains this by stating `Daniel` dropped the milk in sentence 5 and dropped the football in sentence 7. This is an example of pattern **P3**.

For both patterns **P2** and **P3**, tExplain's explanations are a subset of bAbI's, making tExplain's explanations more concise. It's unnecessary to explain the answer on line 6 that `Daniel` is carrying a football by stating he picked up the milk and dropped it.

# Chapter 6

# Conclusion

TEXT2ALM was developed in 2019 and is a successful information extraction system that converts unstructured information present in English narratives into a structured form. TEXPLAIN 1.0 was introduced in 2021 and this extends TEXT2ALM by replacing answer set solver CLINGO with XCLINGO. This replacement allows explanations for extracted information from narratives. In this work, we introduce TEXPLAIN 2.0 that allows automation, queries, updated XCLINGO Trace File, and improved testing on the bAbI dataset. We added the capability to use dropping action verbs as part of explanations in Task 2 (Two Supporting Facts) and achieve 100% accuracy on this task along with Tasks 3 (Three Supporting Facts) and 6 (Yes/ No Questions). We also applied TEXPLAIN on Task 8 (Lists) and recorded our results. TEXPLAIN 2.0 automates many of the parts that would be done by the user in TEXPLAIN 1.0 including logic program creation, narrative preprocessing, and running XCLINGO. These improvements give better usability that allows for further development of the system TEXPLAIN.

# Bibliography

Aguado, F., Cabalar, P., Fandinno, J., Muñiz, B., Pérez, G., and Suárez, F. (2019). A rule-based system for explainable donor-patient matching in liver transplantation. In Bogaerts, B., Erdem, E., Fodor, P., Formisano, A., Ianni, G., Inclezan, D., Vidal, G., Villanueva, A., Vos, M. D., and Yang, F., editors, *Proceedings 35th International Conference on Logic Programming (Technical Communications), ICLP 2019 Technical Communications, Las Cruces, NM, USA, September 20-25, 2019*, volume 306 of *EPTCS*, pages 266–272.

Brewka, G., Eiter, T., and Truszczynski, M. (2011). Answer set programming at a glance. *Commun. ACM*, 54(12):92–103.

Dorsey, A. (2021). Extending text2alm with new relations and explanations by xclingo.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2019). Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82.

Levin, B. (1993). *English verb classes and alternations : a preliminary investigation.* University Of Chicago Press.

Lifschitz, V. (2019). *Answer set programming.* Springer Berlin.

Palmer, M. (2018). VerbNet. `https://verbs.colorado.edu/verb-index/vn3.3/`.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015). Towards AI-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698.