Extending Text2ALM with new relations and explanations by XClingo

## Section I. Introduction

Information extraction (IE) is generally concerned with extracting snippets of meaning from text in natural language and storing the data in structured, machine interpretable form. Text2ALM (Olson and Lierler, 2020) is a fine example of an IE system capable of doing this on simple narratives with so-called action verbs. This work will focus on two ways of extending the Text2ALM approach.

Currently the system is confined to extracting relations that are explicitly present in the sentences of the input narrative or that are present in the description of common sense knowledge in the CoreALM Library -- an integral component of the Text2ALM system. Text2ALM(Olson, Lierler 2019) is built on the backbone of Answer Set Programming (Lifschitz 2002) or ASP --- a knowledge representation declarative programming paradigm with roots in deductive databases. This opens up the possibility of defining new relations to extract further insights from the text.

The programs in ASP consist of rules, whereas solutions to these programs are called answer sets. The system Text2ALM uses these answer sets to construct its structured knowledge base of the facts extracted from text. Generally, when users make decisions based on data extracted from natural language text, they desire to have human understandable explanations to support these decisions. XClingo (Aguado et al. 2019) is an ASP based framework 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 that may make such explanations especially suitable for their goals. We intend to incorporate Xclingo into the Text2ALM framework to construct a new system to provide explanations to derived conclusions/structured facts from given original narratives.

# Preliminaries

## Section II. Review: IE System Text2ALM

Information extraction (IE) is generally concerned with extracting information from unstructured sources such as, for example, natural language text and storing the derived data in structured, machine interpretable form. In this work we examine Text2ALM, a successful example of modern systems capable of performing IE from simple narratives with action verbs. Thus, we first present a brief overview of the major features of Text2ALM through considering a sample input and the resulting output from the system.

Text2ALM takes narratives in natural language as input. Here is an example of such a narrative, which we will refer to as *narrative 1*:

Mary went to the bathroom.

John moved to the hallway.

Mary travelled to the office.

*Narrative 1*

From this narrative the system is able to output files that record facts about this text in structured, machine interpretable form. Let us examine part of the Text2ALM output that consists of facts that we grouped for readability:

*Group 1:*

instance(e1, go\_to), instance(e2, go\_to), 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)

We now provide intuitions behind the readings of the listed facts. For example, the *location* facts provide us with information about the location of entities in the narrative at different points in time. The first group of facts tells us basic information about the three events that happened within the narrative, namely, **e1**, **e2**, and **e3.** For example, event **e1** is an instance of **go\_to** actions. The *event\_agent* and *event\_destination* facts 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 **to\_go** action **e1** and **bathroom** is the destination of **Mary** in the scope of **e1**. Facts, such as *occurs(e1,0),* tell us about the time point associated with the event **e1.** Within the Text2ALM framework each sentence is associated with a discrete time point, wherewe take **0** to denote the time point of the first sentence of the narrative. Each following sentence increments the time point of the previous sentence by one. Consider *location(Mary, bathroom, 1)* in the last group of facts*.* It states that entity **Mary** is located at **bathroom** at the time point **1** associated with the time of completion of the sentence. In addition to these top level facts, Text2ALM encodes more conclusions in the full answer set file. For example, one may infer that all events **e1, e2,** and **e3** are instances of the notion of *move*, or “movement” in plain speech.

It is interesting to note the difference in the nature of the facts in Groups 1-3 versus Group 4. The facts in Groups 1-3 stem directly from text. Text2ALM utilizes a portfolio of natural language processing tools to extract these facts. Yet, to derive facts in Group 4 Text2ALM relies on commonsense knowledge encoded in the CoreALM Library that contains “axioms’’ (background knowledge about behavior) about various kinds of actions. In the scope of our running example axioms about *move* are utilized. Text2ALM uses facts of Group 1- Group 3 together with common sense knowledge about actions as well as the law of inertia (*things normally stay as they are*) to infer information implicitly conveyed by a narrative culminating in the facts of Group 4. This allows the system to answer questions such as “*where was John at the end of the story*”? The answer, of course, is the *hallway* since he moved there in sentence two and there is no information indicating that he otherwise moved upon the completion of sentence 3.

## Section III. Review: Explanations in ASP via Xclingo

The output produced by Text2ALM contains answer sets which are, while easy to understand for those accustomed to viewing ASP, potentially more obscure for the layperson. Xclingo is a system for ASP that allows the programmer to add annotations to logic programs so that generated answer sets are presented together with explanations in a user friendly manner. In the absence of annotations, Xclingo coincides in its workings with the answer set solver Clingo (Gebser, Martin, et al. 2008). We illustrate the use of Xclingo annotations by an example.

Let us consider a simple ASP program inspired by *narrative 1*:

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 follows:

event\_destination(e1,bathroom) event\_agent(e1,mary) instance(e1,move) occurs(e1,0) location(mary,bathroom,1)

In order to define the answer set output as human-readable natural language text we add the following Xclingo annotations to the end of the program, focusing only on the facts/atoms of the form *location(\_,\_,\_)* occurring in an answer set:

%!trace {"% is located at % at time point %",O, D, T} location (O, D, T).

%!show\_trace location(O,D,T).

The annotation works as follows: the directive *%!trace* allows us to associate a *label* with an atom of a program so that whenever an atom of this form occurs in an answer set an explanation due to the associated label is triggered. In our running example, a label has a form of the tuple

{"% is located at % at time point %",O, D, T},

where

* The first element of the tuple is the phrase string in quotations that is meant as an output upon some conditions;
* The % symbols within this string are placeholders for the values of the variables that are listed as the remaining elements of the tuple; For instance, the first occurrence of % will be replaced with the value of O.

An atom with which this label is associated in our sample %!trace annotation has the form *location(O,D,T).*

The %!show\_trace directive of Xclingo instructs the system which atoms of computed answer sets should be displayed. The following is the output of running Xclingo on our sample ASP program with the annotations:

Answer: 1

>> location(mary,bathroom,1) [1]

\*

|\_\_"mary is located at bathroom at time point 1"

As you can see, the sample output from Xclingo is much more immediately understandable to the layperson because it is written in natural language. In contrast, it requires some amount of training and familiarity with answer set programming for an individual to immediately recognize that this information is already encoded in the original ASP program.

# Original Contributions of the Project

## Section IV. Extending IE Systems With New Relations

As it currently stands, system Text2ALM is *confined* to relations that are explicitly present in the sentences of the input narrative or that are present in the description of common sense knowledge in the CoreALM Library. While we gain a plethora of valuable insights from this, we can imagine how to extend it with new relations built on top of the existing relations to gain even more. In particular, in the scope of *narrative 1* we are able to conclude that Mary has been in at least two places. And in order to have been in two places, Mary must have moved from one of them to the other. In the following section, we define this concept precisely and then discuss its encoding within the ASP framework and how it can be used to augment the workings of Text2ALM.

### Vector Edge

In mathematics, a path in a graph is a finite or infinite sequence of distinct edges which joins a sequence of distinct vertices. Therefore, on a graph we can talk about a path along edges from X to Y, where X and Y are vertices. We can apply this concept to analyzing the narrative. In particular, we can talk about how Mary has traveled along a path from the bathroom to the office. A human looking at the narrative would have little trouble understanding, however, what is interesting to us is the potential to glean this utilizing IE system Text2ALM.

Let us see an example of how this could be implemented. We append the following to the ASP program from the previous section inspired by *narrative 1*:

instance(e2,move).

event\_agent(e2,mary).

event\_destination(e2, backyard).

occurs(e2,1).

Answer set of the new program contains two locations for Mary. It is easy to see this running Xclingo as we did in the previous section:

Answer: 1

>> location(mary,bathroom,1) [1]

\*

|\_\_"mary is located at bathroom at time point 1"

>> location(mary,backyard,2) [1]

\*

|\_\_"mary is located at backyard at time point 2"

Here it is easy to see that Mary was in the bathroom at time point 1 and then must have gone to the backyard between then and time point 2 because at time point 2 she is located in the backyard. This means that the location relation can be extended to build the notion of a vector\_edge. The following ASP rules is a way that this could be encoded:

in(O,S,D,B,B+1) :- S != D, location(O, S, B), location(O, D, B+1).

vector\_edge(O,S,D,B,BP) :- in(O,S,D,B,BP), S!=D.

vector\_edge(O,S,D,B,E) :- vector\_edge(O,S,SP,B,BP),

vector\_edge(O,SP,D,EP,E), EP > B, E > B, S!=D.

In mathematical terms, these rules encode that relation *vector\_edge* is a transitive closure of relation *in*. We say that these rules *define* relations *in* and *vector\_edge*.Intuitively we can read them as follows. There is the auxiliary relation “in” which solves as true if source S is not equivalent to destination D, and object O is in two locations such that its first location has source S and time point B, and its second location has destination D and the time point of the previous location plus 1. The relation vector\_edge/5 encodes the notion of an object O transitioning from source location S to destination location D from beginning time point B to end time point E. Following “in”, vector\_edge is encoded as resolving to true if either (i) relation *in* “applies” in the first rule with vector\_edge in its head (the left hand side of symbol “:-”), or (ii) the second rule accounts for cases in which the vector\_edge transition occurs over a time interval greater than 1.

Let us consider the following ASP program inspired by *narrative 1:*

instance(e1,move).

instance(e2,move).

event\_agent(e1,mary).

event\_destination(e1, bathroom).

occurs(e1,0).

event\_agent(e2,mary).

event\_destination(e2, backyard).

occurs(e2,1).

location(mary, backyard, 3).

location(mary, kitchen, 4).

The answer set of this program contains the following atoms of the form *location(\_,\_,\_)*:

location(mary,bathroom,1) location(mary,backyard,2) location(mary,backyard,3) location(mary,kitchen,4) .

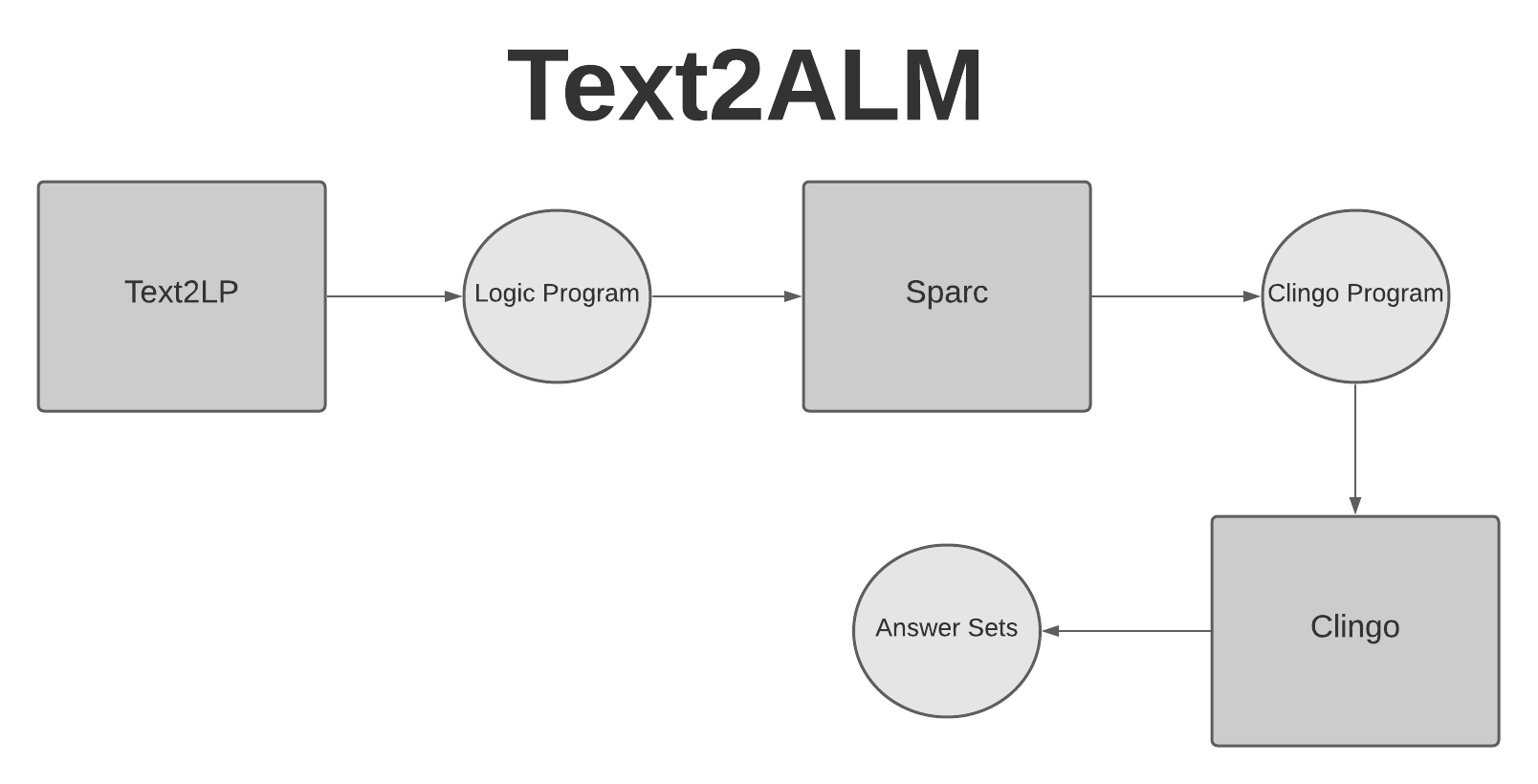
When we extend this program with the rules defining relations *in* and *vector\_edge,* the answer set of resulting program contains the same atoms of the form *location(\_,\_,\_)* as above as well as the following atoms of the form *vector\_edge(\_,\_,\_)*:

vector\_edge(mary,bathroom,backyard,1,2) vector\_edge(mary,backyard,kitchen,3,4) vector\_edge(mary,bathroom,kitchen,1,4).

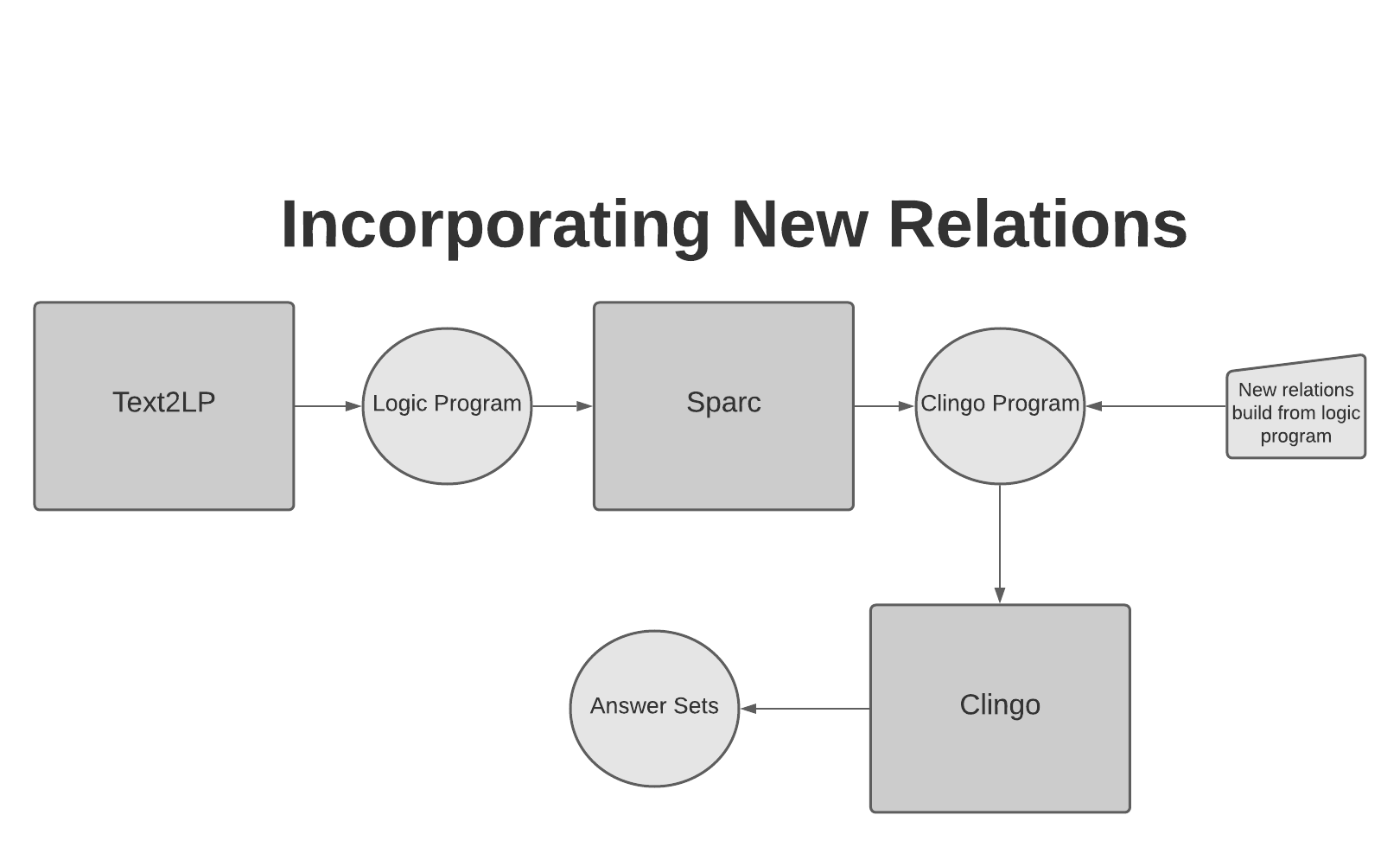
This answer set divulges the information to the reader that object *Mary*

* went from source location *bathroom* to destination location *backyard* from time point 1 to time point 2;
* went from source location *backyard* to destination location *kitchen* from time point 3 to time point 4;
* went from source location *bathroom* to destination location *kitchen* from time point 1 to time point 4.

We now illustrate how system Text2ALM can be extended to incorporate definition of new relations into its architecture. Diagram 1 depicts the key building blocks of original 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 so called background knowledge also in the form of a logic program. Thus the Text2LP translates a given narrative into a logic program in the language of answer set solver Sparc (Balai et al. 2013). The answer set solver Sparc translates this program into the format of answer set solver Clingo and invokes Clingo to compute answer sets of a given program.

**

*Diagram 1*

Diagram 2 proposes a slight change into the Text2ALM architecture where we intercept the output by Sparc. We then extend that output with rules defining new relations. The resulting extended program is then passed on to answer set solver Clingo.   


*Diagram 2*

In the next section, we explore a way to build even further upon what has been described so far.

## Section V. Xclingo Explanations with Text2ALM

### Thus far we have conducted a brief overview of the Text2ALM IE system and the Xclingo explanation system, and we have explored a way to extend relations in Text2ALM in order to form new ones that give more insight. Given the foundation that we have created in the previous sections, we propose a new system that would be a natural extension to Text2ALM by allowing it to rely on the Xclingo framework to produce explanations for extracted information from these relations. We can thus explore a system that provides even more interesting and human readable insights from information extracted from natural language texts. Consider the following narrative in natural language:

John picked up the ball.

John went to the hallway.

Mary traveled to the kitchen.

*Narrative 2*

Running Text2ALM on this narrative results in the answer set that contains the following atoms of the form *location(\_,\_,\_)*:

location(john, hallway, 2), location(john, hallway, 3),

location(ball, hallway, 2), location(ball, hallway, 3),

location(mary, kitchen, 3)

Given the narrative and these atoms in the answer set the user will easily be able to deduce that the ball is in the hallway at the end of the narrative. However, what if we could both make the system’s explanation more easily readable and include an explanation of **why** the ball is in the hallway at the end of the narrative at the same time? In fact, this is possible. A logic program produced by Text2LP can be extended with Xclingo annotations. After that Xclingo is called to compute annotated answer sets. Diagram 3 presents a new architecture of Text2ALM enhanced with explanation capabilities. The primary benefit of incorporating Xclingo into the system is to provide human readable explanations of the answer sets obtained in the system described in Diagram 2.

A logic program produced by the Text2LP component of the Text2ALM system is rather complex. Some considerations are required in order to produce sensible annotations. Below we present a preliminary attempt that serves as a proof of concept for this direction of the research. It is a future work direction to enhance Xclingo annotations for Text2ALM. We showcase our preliminary annotations by following Xclingo trace rules:

%!trace {"% is located at % at time point %", O', D', T} location(O, D, T) : is\_a(O, O'), is\_a(D, D').

%!trace {"since % % the % at time point %", D', X1', B', X2} happened(X1, X2) : is\_a(X1, X1'), is\_a(D, D'), is\_a(B, B'), actions\_vn\_agent(X1,D), actions\_vn\_theme(X1,B).

%!show\_trace location(O,D,T).

The first trace rule encodes an explanation for location which the reader will find familiar with the exception of the *is\_a(\_,\_)* atoms that are also utilized. In a logic program produced by Text2LP, location arguments are encoded not with names such as “ball” or “john,” but with their abstractions/referents such as “r1” or “r2.” However, elsewhere in the program “r1” and “r2” are linked together with “ball” and “john” through the relation *is\_a(\_,\_)*. Therefore, we use the auxiliary *is\_a(O, O’)* together with *O’* instead of *O* inside the curly brackets in order to instruct the trace to display “ball” and “john” instead of “r1” and “r2.”

The second trace rule is then written in order to provide a sub explanation, or justification for location. Our goal is to explain why the ball is located in the hallway at the end of *narrative 2*. Therefore, we want to explain the causes. The relation happened/2 encodes an event with a time point. The rest of 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 actions\_vn\_agent/2 and actions\_vn\_theme/2 in order to associate the entity that commited the event in happened/2 with the recipient of the action of the event in happened/2.

The following is a snippet of the output given by Text2ALM extended with Xclingo when run on *narrative 2.*:

|\_\_"ball is located at hallway at time point 3"

| |\_\_"since john get\_13\_5\_1 the ball at time point 0"

| |\_\_"john is located at hallway at time point 3"

As the reader can see, with these trace rules Xclingo is able to explain to the reader that the ball is located at the hallway at the end of the story (time point 3) because John got the ball at time point 0 and is located at the hallway at time point 3.

### Combining IE System Extensions with Xclingo Explanations

Let us return to the example of vector\_edge/5 and return to the narrative mentioned in section 1:

Mary went to the bathroom.

John moved to the hallway.

Mary travelled to the office.

Running Text2ALM on this narrative produces a rich and extensive answer set program upon which we can add the vector\_edge/5 extension and then a collection of trace rules to produce a human-readable explanation of vector\_edge/5. A convenient way to organize and run this with Xclingo is to place the extension vector\_edge/5 relation and Xclingo trace rules in a helper file which we will call vector-trace.tp.lp

%Inside vector-trace.tp.lp

in(O,S,D,B,B+1) :- instance(O, tangible\_entity), instance(S, spatial\_entity), instance(D, spatial\_entity), timeStep(B), S != D, location(O, S, B), location(O, D, B+1).

vector\_edge(O,S,D,B,BP) :- in(O,S,D,B,BP), S!=D.

vector\_edge(O,S,D,B,E) :- vector\_edge(O,S,SP,B,BP), vector\_edge(O,SP,D,EP,E), EP > B, E > B, S!=D.

vector\_edgeB(O,S,D,B,E) :- vector\_edge(O,S,D,B,E).

%!trace {"% went from % to %", O', S', D'} vector\_edgeB(O, S, D, B, E) : is\_a(O, O'), is\_a(S, S'), is\_a(D, D').

%!trace {"% was at % during sentence %", O', S', B} locationB(O, S, B) : is\_a(O, O'), is\_a(S, S').

%!trace {"% % % at time point %", X0', A, B', X2} happened(X1,X2) : is\_a(X1, X1'), event\_agent(X1, X0), is\_a(X0, X0'), link(X1', A), actions\_vn\_destination(X1,B), is\_a(B, B').

%!trace {"% % % at time point %", B', A, X0', X2} happened(X1,X2) : is\_a(X1, X1'), event\_destination(X1, X0), is\_a(X0, X0'), link(X1', A), actions\_vn\_theme(X1,B), is\_a(B, B').

%!show\_trace vector\_edgeB(O, S, D, B, E).

The logic program extends the native Text2ALM concept of location to define the relation of vector\_edge/5 and then defines the alias vector\_edgeB/5 rule in order to help ensure we can print out more legible explanations. The penultimate section consists of trace rules to define what will be printed out in the final Xclingo explanation. The final section informs Xclingo which particular trace rules to print out, which in our case is the vector\_edgeB/5 alias rule.

Running Xclingo on the logic program output by Text2ALM combined with the helper file results in the following output:

Answer: 1

>> vector\_edgeB(r1,r2,r5,2,3) [1]

\*

|\_\_"mary went from bathroom to office"

| |\_\_"mary was at bathroom during sentence 2"

| | |\_\_"mary was at bathroom during sentence 1"

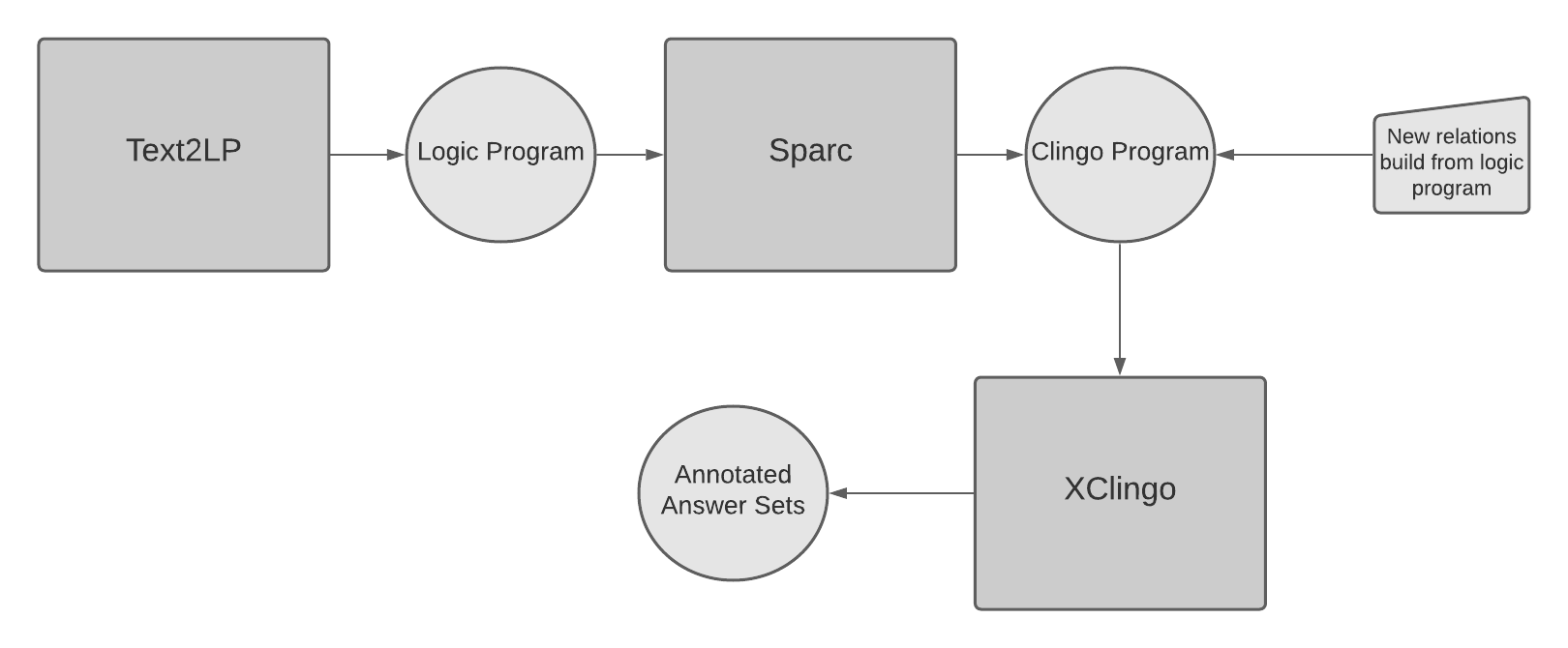
| | | |\_\_"mary move bathroom at time point 0"

| |\_\_"mary was at office during sentence 3"

| | |\_\_"mary move office at time point 2"

This output provides a good explanation of the sole vector edge relation in the original narrative, which is Mary going from the bathroom to the office. The top level explanation simply states that Mary went from the bathroom to the office. From there, the explanation justifies this by explaining that Mary “move” to the bathroom at time point zero, then remained located at the bathroom during sentences 1 and 2. Then it explains that Mary was at the office during sentence 3 because Mary “move” to the office at time point 2.

The diagram below now demonstrates an overview of how this new system would work in its entirety.



*Diagram 3*

The summary of this system in a nutshell is that it receives narratives in natural language as input and its output is an Xclingo explanation file. The Xclingo explanation file provides human readable explanations and justifications of the information contained in the answer sets output from Text2ALM together with answer sets constructed from new relations built upon the answer sets already present in the Text2ALM output.

## Section VI. Methodology

One of the outcomes of this project is the creation of the methodology to guide aforementioned extensions of Text2ALM. In particular, the developed methodology focused on two aspects:

* *how can a new relation be defined to extend Text2ALM*,
* *how can relations be explained using Xclingo* *to produce*  *a sensible, user-friendly output.*

### Defining new relations

First, in order to define a new relation to extend Text2ALM we must consider what information was already present in the current knowledge base and output of the system. Secon, we must consider what other information could be present in the main text. Afterwards, we use the following approach to decide on what to extend:

1. Identify information/conclusions that can be gleaned from the text that aren’t captured in the Text2ALM output.
2. Consider whether this information/conclusion could be of interest and to whom.
3. Define this information/conclusion as a relation and use plain language to describe it.
4. Convert this plain language into logic program rules defining these new relations.

For instance, consider narratives pertaining travelling of people exemplified in our first running example. For this domain the outlined steps of the methodology may result in the following:

1. Identify that the notion of a person or object (which in the code is encoded as a *tangible\_entity*) having followed a path from one place to another was missing.
2. Determine that this information could be of interest to many parties, law enforcement and intelligence services being a few such examples.
3. Define this conclusion, that of a person or object following a path from one place to another, as a vector edge and explain what is meant using plain language and the mathematical notion of a path.
4. Encode vector edge as an arity 5 relation built on top of predefined relations such as location/3.

### Explaining relations

Here we describe the steps to undertake when creating explanations in the proposed system. It is a direction of future work to enhance explanations beyond the current point and refine methodological steps. As noted earlier, system Xclingo is guided through trace rules in producing its explanations. In order to determine what to trace we had to follow some sort of a process. The following is the approach that we took to design of such rules within the scope of Text2ALM:

1. Defined the general idea of what to explain.
2. Identified the relevant top level relation that explains the idea.
3. Composed a trace rule on this top level relation.
4. Identified relevant supporting justifications for the relation.

This process was then repeated on as many levels of the answer set tree as necessary to gain meaningful insights into the narrative. As an example, the process to define the trace rules that resulted in the output from the previous section went like so:

1. We wanted to explain vector\_edge/5 in plain language, i.e., that the object moved from one place to another.
2. We identified that vector\_edge/5 is the top level relation that explains an object moving from one place to another.
3. We composed the trace rule %!trace {"% went from % to %", O', S', D'} vector\_edgeB(O, S, D, B, E) : is\_a(O, O'), is\_a(S, S'), is\_a(D, D').vector\_edge(O, S, D, B, E). to explain this top level relation.
4. We identified that S and D, which respectively stand for Source location and Destination location would be critical supporting justifications for the explanation.

We then simply repeated this thought process on what was then the top level relation, that is, location/3. This process is therefore iterative and repeatable. In fact, in both cases the processes are easily repeatable and it is our hope that future students will be able to use them to build upon the work done here.

## Section VII. Real Time Experimentation

As we have been considering ways in which we could enhance the system to produce more expressive and accurate information extraction, an insight occurred to us that we considered worth pursuing (credit to Dr. William Melanson, March 2021). This is the concept of “real time” when considering narratives in natural language. While at present we consider narratives to occur in linear time which correlates with written sentence order, in reality humans do not always relate narratives in exact chronological order. Let us consider the following narrative to illustrate this:

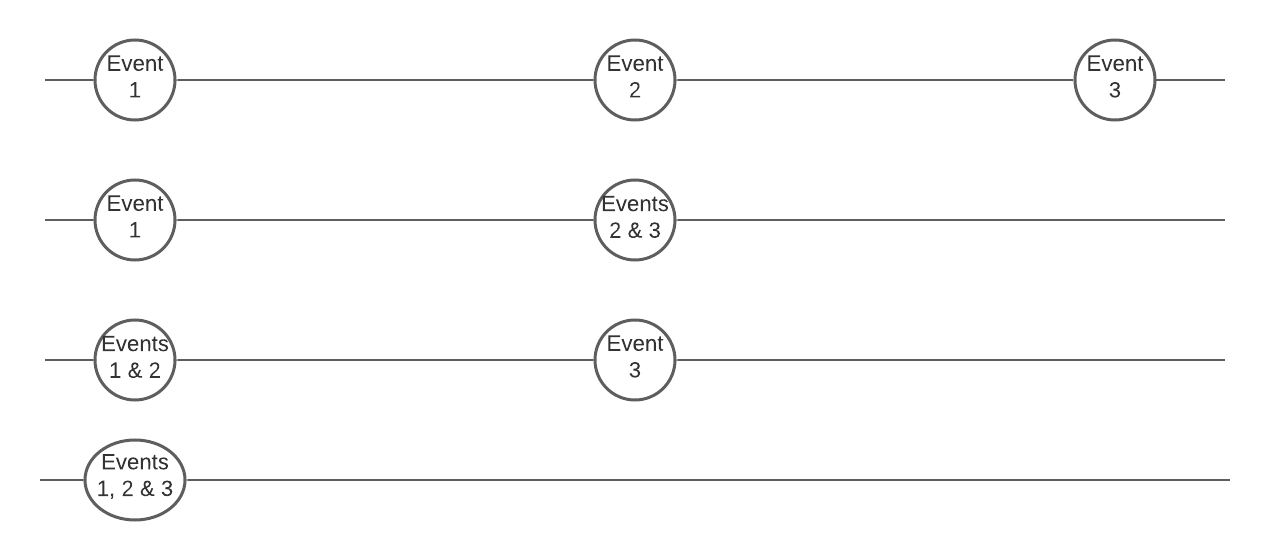
John travelled to the park. 1

Mary went to the hallway. 2

James walked to the kitchen. 3

*Narrative 3*

While it is certainly possible that the events of this narrative occurred in reality in the order in which they were written, one after the other, we consider that in fact this may not be the case. We do not know, in fact, whether Mary went to the hallway after, or at the same time as John travelled to the park. The following diagram provides an overview of the real time possibilities of this narrative.



*Diagram 4*

### Clingo Modeling

A possible solution to this issue that we experimented with is the notion of a *real time counter*. The basic idea of the real time counter is that if two events occur such that their answers sets are: occurs (event1, time1), occurs (event2, time2), if *time2* is greater than *time1*, then *event1* is later than or synchronous with *event2*. Our preliminary results were that we achieved successful small scale modeling with Clingo.

Considering *narrative 3*, we might expect an answer set program to contain answer sets such as the following:

timeStep(1). timeStep(2). timeStep(3). event(travelled). event(went). event(walked). happened(travelled, 1). happened(went, 2). happened(walked, 3).

Therefore, in order to encode our model of Real Time, we made the following encodings:

real\_time(Event, 1):- event(Event), happened(Event, 1).

1{real\_time(Event, RTP); real\_time(Event, RTP+1)}1:- event(Event), happened(Event, T), T>1, happened(PrevEvent,T-1), real\_time(PrevEvent, RTP).

These encodings define real\_time/2 as follows: real\_time/2 has the base case of the event that occurs at timePoint(1), which is the first sentence of the narrative. This is because we want to limit our scope to not consider what may have happened before the start of the narrative. After this point, the head of the second rule of real\_time/2 is encoded as consisting of sets where if there is a set of real\_time/2 such that the *Event* occurs at *timePoint* RTP, or RTP + 1, we take only one such real\_time/2 per answer set. The body of the rule constrains real\_time/2 of an event to be either the *timePoint* of the sentence it is in, or the time point of a previously occurring event. Running clingo on this answer set program with the addition of real\_time/2 rules produces the following four answer sets:

* Answer: 1 real\_time(travelled,1) real\_time(went,2) real\_time(walked,3)
* Answer: 2 real\_time(travelled,1) real\_time(went,2) real\_time(walked,2)
* Answer: 3 real\_time(travelled,1) real\_time(went,1) real\_time(walked,1)
* Answer: 4 real\_time(travelled,1) real\_time(went,1) real\_time(walked,2)

As the reader can see, these answer sets correspond to what is displayed in the diagram 4 illustrations, although answer sets 3 and 4 correspond to the 4th and 3rd line of diagram 4 respectively.

## Section VIII. bAbI Experimentation

The bAbI benchmark (Weston, Jason, et al. 2015) is a Facebook AI Research project that provides a data set of

* narratives in natural language,
* questions about information contained in the narrative, together with
* annotations on which sentences in given narratives contribute to answers to these questions.

We find that this dataset will serve as a good baseline against which the power of the Text2ALM system combined with new relations and Xclingo to reason and answer questions can be assessed.

At the time of this writing, a fair amount of progress has been made towards producing the desired output for simple bAbI experiments. However, more work will need to be done in order to process more complex bAbI narratives. Let us consider the more simple bAbI narratives for which Text2Xclingo satisfies expected results, and in addition the more complex bAbI narratives for which Text2Xclingo still requires more work. We shall also discuss the development of the current trace files used to process the bAbI narratives, the rationale behind them, and the possible next steps in order to achieve the desired results.

### Successful Experimentation

Let us consider the following bAbI example:

1 Mary got the milk there.

2 John moved to the bedroom.

3 Sandra went back to the kitchen.

4 Mary travelled to the hallway.

5 Where is the milk? hallway 1 4

This snippet contains a simple, numbered narrative along with a question at the end. In our preliminary experiments, system Text2ALM extended with an explanation component of Xclingo has been able to satisfy the question by providing the answer conveying information specified by bAbI “hallway 1 4.” Intuitively, “hallway 1 4” refers to milk being in the hallway as a consequence of sentences 1 and 4.

Let us begin by defining what our desired output was. We expected to have the Text2ALM system which incorporates Xclingo to provide explanations similar to the following:

|\_\_"milk is located at hallway immediately after utterance of sentence 4"

| |\_\_"mary got the milk in sentence 1"

| |\_\_"mary is located at hallway in sentence 4"

An explanation such as this would answer the bAbI question with the same amount of information but with the added benefit of producing human readable justifications. We therefore took the following steps in order to attempt to achieve this desired output:

Firstly, we constructed a Xclingo trace file in order to instruct Xclingo to produce that desired output. The trace file was written as follows:

% Part 1: Helper rules to aid readability of output.

locationB(A,B,C) :- location(A,B,C).

top\_concept(move).

top\_concept(obtain).

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).

% Part 2: The trace rules themselves .

%!trace {"% is located at % in sentence %", O', S', B} locationB(O, S, B) : is\_a(O, O'), is\_a(S, S').

%!trace {"% %d to % in sentence %", X0', A, B', X2 + 1} happened(X1,X2) : is\_a(X1, X1'), event\_agent(X1, X0), is\_a(X0, X0'), event\_top\_concept(X1', A), actions\_vn\_destination(X1,B), is\_a(B, B').

%!trace {"% %d to % in sentence %", X0', A, B', X2 + 1} happened(X1,X2) : is\_a(X1, X1'), event\_agent(X1, X0), is\_a(X0, X0'), event\_top\_concept(X1', A), actions\_vn\_location(X1,B), is\_a(B, B').

%!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').

% Part3: The restriction on which trace to show.

%!show\_trace locationB(O\_G,D\_G,VAR\_0).

*Figure 4: Trace Rules Base*

The trace rules file consists of three parts. The first part of the file consists of “helper rules” which help to ensure that the Xclingo output produces human readable text. We will now explain the role of its elements.

To process bAbI narrative-question pairs, the occurrence of atoms whose predicate is “location/3” (or “location/3”-atoms for short) in an answer set is key. For example, to conclude that "milk is located at hallway immediately after utterance of sentence 4", an atom of the form

“location(milk\_referent, hallway\_referent, 3)”

is expected to be part of an answer set found by Text2ALM system processing a given narrative (where milk\_referent and hallway\_referent stand for the names of the constant symbols assigned by Text2ALM to the entities of ‘milk’ and ‘hallway’ of the given narrative). The occurrence of these atoms provides us with the information on where milk is located at a certain time point. However, we are also interested in explaining why we derive this conclusion. For this purpose, we define the auxiliary predicate “locationB/3”, which can be seen as an alternative/synonymous name to the predicate “location/3”, i.e., relation “locationB/3” holds between elements if and only if relation “location/3” holds between the same elements. We then instruct Xclingo to provide us with the explanations for occurrences (in an answer set) of “locationB/3”-atoms. A natural question to ask is why we introduce the auxiliary relation “locationB/3” to begin with. Why do we not directly ask Xclingo to explain “location/3” atoms? The reason is because logic rules encoding background knowledge about interactions of actions rely on the relation “location/3” in a multitude of ways. Thus, it often happens that “location/3”-atoms are “explained” in terms of other “location/3”-atoms. Formally, the Xclingo system would then be completing a chain of recursive explanations for each occurrence of “location/3”-atoms in a constructed explanation. However, the level of detail of such an explanation is beyond what a human finds natural. Thus, introducing the “locationB/3” predicate allows us to bypass such verbose explanations.

We define two “top\_concept/1”s, ‘move’ and ‘obtain’. Then we define “link\_r/2” and finally “event\_top\_concept/2”, which itself is 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, *Mary getting the milk*, are defined using VerbNet terms, in this case ‘get\_13\_5\_1\_1’. However, ‘get\_13\_5\_1\_1’ 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 ‘get\_13\_5\_1\_1’ up to our “top\_concept/1” of ‘obtain.’ The same applies to the event of *Mary travelling to the hallway*. Immediately in the Text2ALM answer set program, the verb associated with this event is ‘run\_51\_3\_2\_1’, which we propagate up to our “top\_concept/1” of ‘move’.

The second part of the trace rule file consists of the traces themselves. These trace rules define what will be output by Xclingo. For a detailed explanation of how trace rules work in general, we refer the reader back to Section V. We structured the trace rules in this fashion in order to capture a coherent, hierarchical explanation.

The reader may note that trace rules 2 and 3 begin identically, however, we have included both trace rules because in our narrative, when it comes to the event of moving there are two slightly different paths that lead to our “top\_concept/1” of ‘move.’ One path utilizes “actions\_vn\_destination/2” whereas the other utilizes “actions\_vn\_location/2.” Finally, the third part of the trace file simply instructs Xclingo on which rule to show the explanations for, which in our case was “locationB/3”.

After constructing this trace file, we directly processed it, along with the answer set program output by Text2ALM, utilizing Xclingo. The following is the command we used to run it:

**python3 xclingo.py milk\_narrative.tp.lp milk-trace.lp**

Here we use the xclingo.py file to process the answer set program output by Text2ALM, milk\_narrative.tp.lp, along with the trace rules, placed in the file named milk-trace.lp. The output of this command successfully achieved the goal of being similar to the expected output mentioned above:

|\_\_"milk is located at hallway in sentence 4"

| |\_\_"Since mary obtained the milk in sentence 1"

| |\_\_"mary moved to hallway in sentence 4"

Finally, we then tested the entire system Text2ALM combined with Xclingo explanations at once to ensure successful integration. We utilized the following command in order to do so:

**./xclingo.sh milk\_narrative.txt milk-trace.lp**

Our results were the same, indicating successful integration.

Admittedly, we were not able to simply use the word ‘got’ and instead the output was ‘obtained’. In order to arrive at the output of ‘got’, future work potentially might employ a post-processing script, or perhaps a change in internal processes to Text2ALM. Additionally, instead of the third sentence saying “mary is located at hallway in sentence 4”, we said “mary moved to hallway in sentence 4.” This output would require similar processes to achieve, however, it may be more complex because of the explanation hierarchies defined in the internal processes of Text2ALM, namely, SPARC. “However, apart from these small discrepancies, our output essentially delivered on producing human-readable explanations that supply the desired information to answer the bAbI challenge. We answer the question of where the milk is, and then we reference sentences 1 and 4 as justifications, with the added benefit of explanations in plain English.

### Experimentation Limitations

Let us now consider the limitations of the current system with a longer bAbI example snippets:

Mary moved to the bathroom.

John went to the hallway.

Daniel went back to the hallway.

Sandra moved to the garden.

John moved to the office.

Sandra journeyed to the bathroom.

Mary moved to the hallway.

Daniel travelled to the office.

John went back to the garden.

John moved to the bedroom.

This snippet contains a simple, numbered narrative along with 5 questions peppered throughout it. In our preliminary experiments, system Text2ALM extended with an explanation component of Xclingo has been able to satisfy all of the questions with the addition of providing human-readable explanations, but only with manual intervention. The bAbI benchmark asks 5 questions about the location of the people in this 10 sentence narrative. The questions are formatted like so: “Where is <person>”? Intuitively, the answers, such as the answer to the first question, “where is Mary?”, which the benchmark bAbI lists as “bathroom 1”, refers to Mary being in the bathroom as a consequence of sentence 1. Therefore, our goal was to have our system output these answers along with human readable justifications for them.

Let us begin by specifying what our desired output was. We expected to have the Text2ALM system which incorporates Xclingo to provide explanations similar to the following:

|\_\_"mary is at bathroom"

| |\_\_"mary moved to bathroom during sentence 1"

An explanation such as this would answer the bAbI question with the same amount of information but with the added benefit of producing human readable justifications. We therefore took the following steps in order to attempt to achieve this desired output:

We attempted to process this narrative utilizing the same trace rules base referred to in the successful experimentation.

**python3 xclingo.py question1.tp.lp question1-trace.lp**

Here we use the xclingo.py file to process the answer set program output by Text2ALM, question1.tp.lp, along with the trace rules, placed in the file named question1-trace.lp.The output of this command produced some correct explanations, but also some explanations that were more verbose than expected with respect to the goal of being similar to the expected output mentioned above. It did correctly produce output for the first question:

|\_\_"mary is at bathroom during sentence 1"

| |\_\_"mary moved to bathroom during sentence 1"

However, as an example, the answer to “Where is Daniel?” is as follows:

|\_\_"daniel is at office during sentence 8"

| |\_\_"daniel moved to office during sentence 8"

| |\_\_"daniel moved to hallway during sentence 3"

This explanation contains extraneous information, namely, the information about Daniel moving to the hallway during sentence 3. It was determined that the following rule in the answer set program produced by Text2ALM to capture the meaning of the given narrative was necessary to modify:

location(X0\_G,X1\_G,VAR\_0):-location(X0\_G,X1\_G,I\_G),

not -location(X0\_G,X1\_G,VAR\_1), **dom\_location(X0\_G,X1\_G,VAR\_2)**,VAR\_0=I\_G+1,VAR\_1=I\_G+1,

VAR\_2=I\_G+1, spatial\_entity(X1\_G),timeStep(VAR\_0),timeStep(VAR\_2),

timeStep(I\_G), tangible\_entity(X0\_G),timeStep(VAR\_1).

In particular, the atom

“**dom\_location(X0\_G,X1\_G,VAR\_2)”**

was replaced by literal

**“not not dom\_location(X0\_G,X1\_G,VAR\_2)”**

in this rule. This change does not change the meaning/semantics of the overall program but is crucial in directing Xclingo towards the kinds of explanations we are looking for. In particular it suggests to Xclingo to avoid “tracing” explanations for “dom\_location/3”-atoms. Indeed, these atoms are present in a program merely to encode sort information of participating entities. We note that at the moment Xclingo developers are incorporating a new feature into Xclingo, where one may specify a list of predicate names whose atoms will be avoided in explanation-constructions (this project was an inspiration to such a development). This feature will prove to be of value in this application and we believe it is general enough to be of use in other applications.

After this modification in **question1.tp.lp**, the output produced by the command line

**python3 xclingo.py question1.tp.lp question1-trace.lp**

included our expected output

|\_\_"daniel is at office during sentence 8"

| |\_\_"daniel moved to office during sentence 8"

## Section IX. Future Work

Moving forward, the next step of work on this project is to automate more parts of the system to facilitate the engineer’s ability to concentrate on higher level abstractions rather than configuration details. As it currently stands, engineers much concern themselves with having a relatively extensive amount of knowledge about python version control and must go through a fairly complex installation process in order to set the system up. It would be preferable to develop some sort of automated version control system to ensure minimal breakage in the components that rely on python and to streamline the installation process to reduce the number of steps involved in it.

Additionally, advancement in the maturity of this Text2ALM with Xclingo system is a goal of future work. For instance, there is still much exploration and coverage of the bAbI benchmark that remains to be done. It is apparent that this will be a non-trivial task, as a deep understanding of the inner workings of Text2ALM will be necessary in order to modify the system to work with Xclingo to successfully meet the entire bAbI benchmark. Furthermore, it will require cooperation with the Xclingo team as it continues to develop the capacity of Xclingo to quickly process large inputs.

Finally, when it comes to real time, while we were able successfully produce a small scale Clingo model of our notion of real time, there are some considerations which we would have to address when applying this concept. For example, we would have to consider the case of the same actor doing multiple actions at once. Consider the phrase “Jack is walking while talking on his cell phone.” These are two distinct actions, but they occur at the same time. However, at other times, the same actor might take two actions which must occur one after the other. Therefore, restricting real time on the level of an actor would have to be specific to a given action. There are several other considerations, currently known or otherwise, that would have to be taken into account in order to produce a reasonably elaboration tolerant concept of real time.

## Section X. Conclusion

Information extraction systems capable of processing natural language offer promising potential benefits to a wide range of public and private institutions, as well as society at large. Text2ALM is a successful IE system that has shown the capability of extracting useful insights using a predetermined set of relations. The system is limited, however, to information explicitly stated in the narrative text. Yet, since the output is structured in a way that imitates such data as that which is found in relational databases, it becomes apparent that it can be extended and aggregated to form new relations. Xclingo is a powerful explanation engine that is capable of annotating answer set programs in order to output human readable explanations in natural language. The proposed system in this project takes the extended relations built from Text2ALM output and combines them with Xclingo in order to provide us with more insights into text in natural language. The example provided in this technical report took a simple narrative and was able to conclude from the it that Mary traveled from the bathroom to the office even without that information being explicitly stated in the narrative. This system’s ability to thus imitate human reasoning offers an exciting and promising new direction of investigation and development.

## Section XI. Bibliography

Lifschitz, Vladimir. "Answer set programming and plan generation." *Artificial Intelligence* 138.1-2 (2002): 39-54.

Aguado, Felicidad & Cabalar, Pedro & Fandinno, Jorge & Muñiz, Brais & Pérez, Gilberto & Suárez, Francisco. (2019). A Rule-Based System for Explainable Donor-Patient Matching in Liver Transplantation. Electronic Proceedings in Theoretical Computer Science. 306. 266-272. 10.4204/EPTCS.306.31.

Olson, Craig & Lierler, Yuliya. (2019). Information Extraction Tool Text2ALM: From Narratives to Action Language System Descriptions. Electronic Proceedings in Theoretical Computer Science. 306. 87-100. 10.4204/EPTCS.306.16.

Balai, Evgenii, Michael Gelfond, and Yuanlin Zhang. "Towards answer set programming with sorts." *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, Berlin, Heidelberg, 2013.

Gebser, Martin, et al. "A user’s guide to gringo, clasp, clingo, and iclingo." (2008).

Karin Kipper-Schuler (2005): VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon. Ph.D. thesis, University of Pennsylvania.

Weston, Jason, et al. "Towards ai-complete question answering: A set of prerequisite toy tasks." *arXiv preprint arXiv:1502.05698* (2015).