Deriving Time Lines from Texts

Mathias Landhäußer
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
landhaeusser@kit.edu

Tobias Hey
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
tobias.hey@student.kit.edu

Walter F. Tichy
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
walter.tichy@kit.edu

ABSTRACT

We investigate natural language as an alternative to programming languages. Natural language would empower anyone to program with minimal training. In this paper, we solve an ordering problem that arises in natural-language programming. An emprical study showed that users do not always provide the strict sequential order of steps needed for execution on a computer. Instead, temporal expressions involving "before", "after", "while", "at the end", and others are used to indicate an order other than the textual one. We present an analysis that extracts the intended time line by exploiting temporal clues. The technique is analyzed in the context of Alice, a 3D programming environment, and AliceNLP, a system for programming Alice in ordinary English. Extracting temporal order could also be useful for analyzing reports, question answering, help desk requests, and big data applications.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environments; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Language parsing and understanding*

General Terms

Languages, Theory

Keywords

Natural language processing, time lines, temporal expressions, temporal reasoning, Alice, programming with natural language, end-user programming

1. INTRODUCTION

We investigate the use of natural language as a substitute for programming languages. Human beings are for the most part adapt at instructing – they teach children, train apprentices, or direct subordinates. Instructing in natural language is easy for humans, whereas programming languages are difficult to master. Only a tiny fraction of the population can cope with artifical programming languages. Natural language, however, would empower anyone to program. Being able to program is especially desirable now that nearly everybody owns programmable devices in the form of phones, tablets, laptops, or PCs. Since ordinary language comes easy for most people, it might also be the medium of choice for instructing computers. However, achieving this vision is a formidable challenge with many problems to be solved.

In this paper, we address one of the sub-problems, the problem of non-sequentiality: humans do not necessarily enumerate computational steps in the order they need to be executed. Instead, they provide a narrative in which the important aspects are emphasized. Additional steps are filled in out-of-order using temporal expressions such as "before that, do ...", "after a, do b", "at the same time do ...", etc. Thus, extracting the total order of steps is a problem that needs to be solved in order to allow for natural expression of intent.

We discovered the problem of non-sequentiality when asking users to write scripts in natural language. The scripts were meant to describe animations in Alice [2]. Alice is a 3D animation environment intended to teach children programming in a dialect of Java and is developed at Carnegie Mellon University (CMU). Our aim is to replace Alice's Java with ordinary English. We are in the process of building AliceNLP, a package that generates Alice programs from natural language scripts. When working with real scripts, the generated code sometimes turned out to be incorrect because AliceNLP did not interpret temporal expressions properly. Asking users to provide the correct order as in a programming language would be unnatural for them. To support this aspect of naturalness, we needed to solve the problem of non-sequentiality.

Roughly, our temporal rectification works as follows. Initially, AliceNLP enumerates individual steps in textual order. This order is then modified by interpreting clues given by temporal adverbs and prepositions. The clues are translated into a set of ordering operators, which are then applied to the original sequence. An evaluation with 24 scripts describing three Alice animations showed that 86% of the temporal clues are correctly detected and the correct time line was established in 17 of the 24 scripts. At most two reorderings were missing in the incorrect time lines.

Section 2 introduces AliceNLP and Section 3 reviews related work. The following sections present our treatment of temporal expressions and its evaluation.

2. THE ALICENLP PROJECT

AliceNLP is an environment in which users program in natural language. As other projects before (c.f. Section 3), we chose a restricted domain to work with: CMU's Alice. Alice is a programming framework for teaching object-oriented programming to children [2]. It provides a library of 3D objects (humans, animals, vehicles, backgrounds, and so forth) that are used for building and animating a "world". An animation is a Java-like script composed by using drag-and-drop. Eventually, the learner is expected to switch from drag-and-drop to writing code. AliceNLP adds ordinary English as programming language.

Our approach is not limited to Alice. AliceNLP's tools internally use an ontology that describes the target library; by replacing the ontology, a different library could be enabled. The only Alice-dependent components are the ontology creator and the Alice script generator. We use WordNet [7] to deal with synonyms and generalizations, so users have a wide range of expression. At this time, the only requirement is that the library must use descriptive names, as coding conventions suggest. We believe that our approach is indeed portable to many different end-user programming environments.

2.1 Empirical Study

AliceNLP aims at understanding what the end-user wants, not at teaching a particular programming language. At the outset of AliceNLP, we needed to know how humans would program in natural language. To that end, we asked subjects with and without programming backgrounds to describe short Alice animations in their own words. We provided an example animation like the one in Figure 1 together with a sample description. We explained our project's goal and the way subjects were supposed to produce the description. They were told to watch the animation closely and then to write down the description; pausing and re-playing was allowed. We provided the vocabulary for the objects and their capabilities and asked the subjects to describe the animation as precisely as possible, not to leave out anything, to stick to the given names, and to describe the actions in the correct order. We provided hints such as "Start with the background and do not forget to tell the system where the objects are in the beginning. Then describe the animation step by step." We encouraged them to check their descriptions after they wrote it.

Some subjects used text editors, others used pen and paper; we removed spelling errors and minor grammatical issues (e.g., using "him" when referring to an animal) but left the texts untouched otherwise. The texts form a growing corpus of animation descriptions that drive our research. The corpus is also used for evaluation purposes.

Initially, we obtained 15 different texts describing a single animation: seven written by authors without any programming knowledge and eight by authors with programming skills. Even though we thought the task of describing animations not overly complex, we identified the following challenges:

- Level of abstraction: Some authors expected too much from the system and omited actions (e.g., opening a box before looking into it); other authors included too much detail (e.g., the look of a person).
- Use of synonyms: Subjects did not stick to the vocabulary provided. Instead, they used multiple synonyms (for example, "bunny" in one sentence and "rabbit" in the next).
- Incremental scene setup: Some authors described the initial scene only partially and added objects on-the-fly as they needed them for the animation's actions.
- Non-sequentiality: Even though we asked for a step-by-step description, authors often ignored that. Instead, they used temporal expressions to shape the story line.

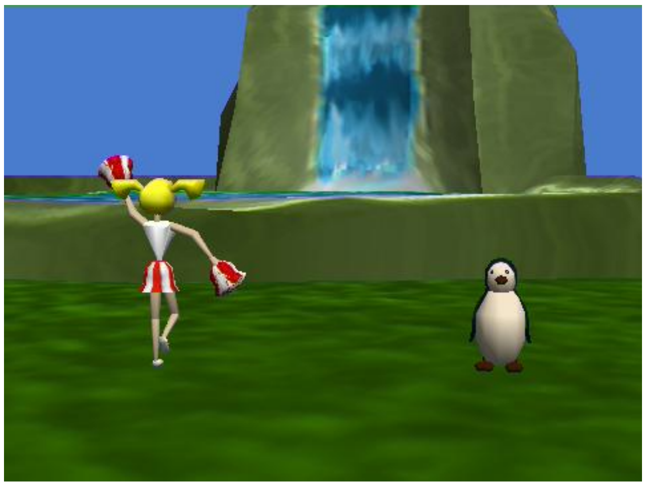


Figure 1: Screenshot of an Alice animation.

In this paper, we concentrate on the challenge of non-sequentiality. If the animations are translated for execution on a computer, all animation actions must be strictly ordered. In order to obtain more samples, we designed new animations and asked additional subjects to write descriptions for them. We particularly asked for non-sequential stories. One subject wrote rather convoluted ones. More about this case in the evaluation.

Our corpus currently contains ten animations and two static scenes with several descriptions each; it is available on the project's web site[1].

2.2 Running Example

Figure 2a shows a simple description of an animation in natural language; this example will be used throughout the paper. As shown in Figure 1 there are a cheerleader and a penguin. At the beginning, the cheerleader cheers. This frightens the penguin, which then glides away.

The text in Figure 2a is correctly ordered. The text consists of one sentence per action. The labels in parenthesis are not part of the actual description. The adverb "twice" triggers a loop. Alice also needs a scene setup (Figure 1). The text for the setup is excluded for brevity as this paper focuses on the dynamic part of the script.

Figure 2b describes the same animation as Figure 2a, but not in sequential order. The actions are marked with the same labels as before.

To represent the temporal order of actions we use time lines. Figure 3a shows the timeline for the sequential de-

[1]`https://svn.ipd.kit.edu/trac/AliceNLP`

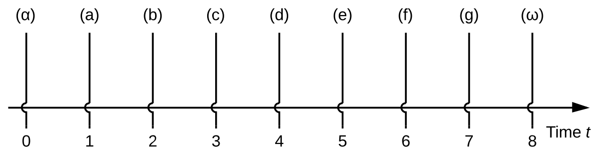
The cheerleader cheers (a). The penguin flaps (b) its wings twice. The cheerleader turns (c) to face the penguin. The penguin turns (d) its head right. The penguin flaps (e) its wings once. The penguin glides (f) away. Then the cheerleader says (g): "Where are you going?".

(a) Sequential description

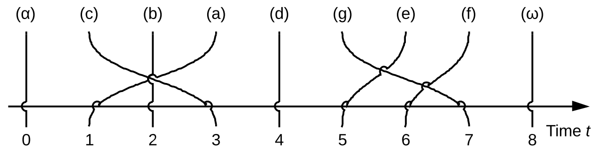
Before the cheerleader turns (c) to face the penguin, the penguin flaps (b) its wings twice. But at the beginning of the scene, the cheerleader cheers (a). After the cheerleader turns (→c) to face the penguin, the penguin turns (d) its head right. At the end, the cheerleader says (g): "Where are you going?". But before that the penguin flaps (e) its wings once and then glides (f) away.

(b) Non-sequential description.

Figure 2: Example descriptions.



(a) Time line for the description in Figure 2a



(b) Time line for the description in Figure 2b

Figure 3: Time lines for the example animation.

scription; it is [α, a, b , c, d, e, f, g, ω], where the roman letters are action labels and α and ω are empty actions that mark the ends of the time line. Action a, which is the cheerleader's action "cheer", is the first proper action in the time line. The non-sequential description has a different time line to begin with. It is shown in the top of Figure 3b. The curved lines indicate the reordering necessary to bring it into the same order as the sequential example. Recall that actions are labeled the same way in both descriptions.

3. RELATED WORK

Research in natural language understanding deals with detecting events and putting them on a time line. Pustejovsky et al. describe an approach that concentrates on exact points in time [14]. They analyze newspaper articles to build a question answering system that can handle queries such as *What happened in French politics last week?* The analysis is aware of temporal expressions such as *last week* and can resolve relative time specifications in the context of the article (e.g., publication date, context of the news, referrals to other events, and so on). The evaluation uses the TimeBank corpus [12] which is annotated with TimeML [13]. TimeML annotates events and points in time; links between different annotations form a temporal order of events. Absolute points of time do not occur in our samples. Indeed, programs, including animations, are expected to be repeated at arbitrary points.

Schilder analyzes legal documents to create temporal constraints for events, e.g., *entered the USA before December 31, 2005* [17]. After the analysis one knows that the event *entering* happened before Dec. 31, 2005 but one cannot pinpoint the exact point of time. If the document states that the person *entered the USA on Dec. 31, 2005*, Schilder can extract the exact point of time and anchor the event on the global time line. Again, absolute points on the time line are not yet needed in our context.

Ohlbach's system for Computational Treatment of Temporal Notions (CTTN) focuses on terms that describe temporal notions such as *noon*, *tomorrow*, and *every Tuesday* [9]. CTTN translates natural language into numeric points in time, ranges, and time lines, but the notions modeled do not occur in our scripts.

There is a body of research on temporal reasoning [16]. For example, Russell et al. describe an event calculus [15]. The emphasis in this work is on automated reasoning using the calculus, not the extraction of event order from texts. Our analysis could provide input for temporal reasoning.

Regarding programming in natural language, Pane and Myers investigated how non-programmers describe programming solutions to make future programming languages more user-centered [8][10]. Ballard and Biermann present an approach for natural language programming of matrix operations called NLC [1]. It allows users to input data and manipulate matrix entries using English. User interactions are handled strictly in sequential order.

Price et al. present NaturalJava, a prototypical user interface based on natural language for creating, modifying, and examining Java programs [11]. One can enter a Java program using this system. Essentially, it is a Java dictation system. The problem of non-sequentiality is not addressed. An important difference is that AliceNLP users do not need to write Java code at all.

Liu and Lieberman perform feasibility studies for programming in natural language [4]. In paper [6] Mihalcea, Liu, and Lieberman describe how programming steps, loops, and comments can be identified in natural language and mapped to programming constructs. Liu's and Liberman's tool Metafor generates program skeletons from stories [5]. No executable code is generated, so the problem of order does not arise.

Knöll and Menzini research an approach to natural language programming called Pegasus [3]. Pegasus translates natural language input in an intermediate "idea language", which is then compiled into real code. The problem of deviations from sequential order is not addressed.

In summary, advances in temporal reasoning may become applicable at a later stage of our work. Natural language processing can produce absolute time lines given exact dates and times; again, this work may be needed later for natural language programs dealing with dates and times. As far as we can tell, research in natural language processing has up to now not dealt with deviations from strict sequential order in realistic scripts produced by humans.

4. ANALYZING TEMPORAL EXPRESSIONS

As mentioned before, people do not always describe action sequences in strictly sequential order. When they deviate,

Table 1: Signal words for temporal expressions

Temporal Adverbs	Temporal Prepositions
after, afterward(s), afterwhile, before, beforehand, concurrently, eventually, finally, first (of all), firstly, henceforth, hereafter, hereupon, initially, last, later (on), simultaneously, subsequently, synchronously, then, thereafter, thereon, thereupon, ultimately, when, whereupon, while, plus some archaic forms	after, as, at, before, by, in, prior to, subsequent

they use temporal expressions that indicate where in the sequence a given action is to be placed. The temporal expressions involve *before*, *at the beginning*, *after*, and others. Temporal expressions modify the textual order.

4.1 Temporal Expressions

We gathered temporal expressions (TEs) in the English language. TE is the umbrella term for the three types of expressions in English which describe temporal relations: Tense, temporal adverbs and temporal prepositions.

Tense alone is not a useful indicator. This is because tense by itself is not able to define a temporal relation. A temporal relation could be established if tense changes from one part of the text to another. However, the tense in the analyzed texts changes rarely, and if so, incorrectly. Action descriptions are best written in the present tense.

The other two types are useful for establishing temporal order. Temporal adverbs include words such as *afterwards*, *yesterday* and *now*. Temporal prepositions such as *ago* and *before* form temporal relations between actions. To form our temporal patterns, we use the signal words shown in Table 1. We omitted terms such as *now*, *yesterday* or *ago* as they are irrelevant for action descriptions. We are also excluding simultaneous or parallel actions for the time being.

4.2 Patterns for Temporal Expressions

Signal words are used in certain patterns to form temporal relations. Every pattern translates to a temporal operator and refers to two actions. There are three operators: $after(a, b)$, $before(a, b)$, and $at(n, a)$: For the first two operators, both parameters are actions; the first one is called *anchor action*, the second one *transfer action*. $after(a, b)$ places the action b directly after the action a; $before(a, b)$ places the action b directly before the action a. Note that $after(a, b)$ is not the same as $before(b, a)$ because the transfer action is moved by the operator, whereas the anchor action keeps its position on the time line. For example, the operator $after(a, c)$ transforms the time line [α, a, b, c, ω] to [α, a, c, b, ω] whereas $before(c, a)$ results in [α, b, a, c, ω]. The operator $at(n, a)$ inserts the action a at the (numerical) position n of the time line.

The position of a TE in a sentence determines the anchor and transfer actions. In *before a, do b* the anchor action is a and b is the transfer action: before(a, b). If one places *before* between the first and the second action, the effect is reversed: *Do a before b* is interpreted as before(b, a). A

Table 2: Temporal Patterns and Operators.

Temporal Pattern	Operator
At the end, do a.	$before(\omega, a)$
In the beginning, do a.	$after(\alpha, a)$
As n^{th} do a.	$at(n, a)$
After a, do b.	$after(a, b)$
Before a, do b.	$before(a, b)$
Do a. *Before that*, do b.	$before(a, b)$
Before a *started*, do b.	$before(a, b)$
Do a. *Then* do b.	$after(a, b)$
When a, do b.	$at(a, b)$
...	

TE which is used in a subordinate clause behaves like a TE placed in a sentence which only consists of this subordinate clause.

If a TE is followed by a word like *that*, this word has an impact on the references if it forms an extended TE; the extended TE has a different temporal meaning than its base: For example, *before that* forms a different temporal relation than *before*.

Our analysis uses TE patterns. We identified over 20 such patterns (not counting synonyms) and Table 2 shows an excerpt of our list.

4.3 Process

The actual approach of identifying the changes of the temporal order of a text is illustrated in Figure 4. First the text is processed with the Stanford parser [18], which provides part of speech tags, a phrase-structure tree, and grammatical dependencies between the words. Afterwards the time line is naïvely initialized, i.e. we arrange the actions in textual order. Then the text is analyzed sentence-by-sentence to identify the TEs and incorporate the changes of the temporal order. The process includes a signal word search, the identification of temporal patterns, a check if a TE is in a subordinate clause, the determination of the anchor and transfer actions, and the update of the time line corresponding to the identified changes.

Matching Temporal Patterns.

AliceNLP searches for signal words in the sentence and checks if they form one of the relevant TE. The particular TE is important, because each expression forms its temporal relation in a slightly different way. AliceNLP also determines the position of the TE relative to the actions in the sentence: It is important to know whether the TE precedes the actions or is located between them (e.g., "*before* a, do b" and "do a *before* b"), because that influences which action is to be moved and where it has to be moved. A pattern consists of a TE, an anchor action and a transfer action.

Splitting Subordinate Clauses.

AliceNLP uses the position of the TE in the sentence to determine whether to ignore parts of the sentence: If the expression is in a subordinate clause, the sentence can be split at the conjunction; only the part containing the TE must be analyzed further. If both parts contain a TE, the analysis proceeds in both parts separately.

Determining Anchor Action and Transfer Actions.

The anchor action can be stated in three different ways:

1. Absolutely with expressions like *at the end* or relatively with expressions such as *before that*.
2. As an action in the same sentence: *Do a after b*.
3. As a reference to an action in a previous sentence: *Do a. After a do b*.

One can handle the first two cases with the information gained during the pattern search alone. The third case is more complex because the referenced action must be determined first. Since the formulation of such a reference can vary greatly and there is no off-the-shelf co-reference analysis for actions, we concentrated on references that literally repeat the action. Such a reference is used in the third sentence of the non-sequential example: "After the cheerleader turns to face the penguin..." Here, the "turn" clause is a repetition of a clause in the previous sentence (marked with $\rightarrow c$ in Figure 2b).

Rearranging the Time Line.

The effect of the TE is encoded for every pattern; Table 2 indicates where the transfer action should be placed.

If no TE can be found, no rearrangements occur. After all sentences are analyzed, the resulting time line represents the recognized temporal order of the actions in the text. At the moment, TEs that demand simultaneous actions are not included in our implementation; extending the patterns appears easy and parallel actions could be included in the time line with fork and join operators.

4.4 Reordering by Example

We demonstrate our approach with the example in Figure 2b. For this purpose we assume the first three sentences to be fully processed and the actions a-d to be correctly ordered on the time line: [α, a, b, c, d, g, e, f, ω].

We begin the analysis with the sentence *At the end, the cheerleader says: "Where are you going?"*. At first, AliceNLP searches for temporal patterns and identifies *at the end* as the only TE in the sentence. It also detects that the TE is at the beginning of the sentence and that the sentence contains only one action: *says (g)*. Then it checks whether the TE is in a subordinate clause (which is not the case). Now it determines the anchor and transfer actions: The TE *at the end* with only one action has no ambiguous references, because *at the end* always refers to the final position of the time line; therefore the anchor action is ω. Since *says* is the only action, it has to be the transfer action. With this information AliceNLP can perform the actual rearrangement of the actions on the time line: $before(\omega, g)$. The resulting time line is [α, a, b, c, d, e, f, g, ω].

Next, the process analyzes the sentence: *But before that the penguin flaps its wings once and then glides away.* Again AliceNLP searches for temporal patterns and identifies the two TEs *before that* and *then.* The expression *before that* is at the beginning of the main clause and the expression *then* is at the beginning of the subordinate clause. AliceNLP splits the sentence at the conjunction (*and*) and processes both parts separately.

The first part *But before that the penguin flaps its wings once* contains only the action *flaps (e)* which therefore is the transfer action of *before that. Before that* always references

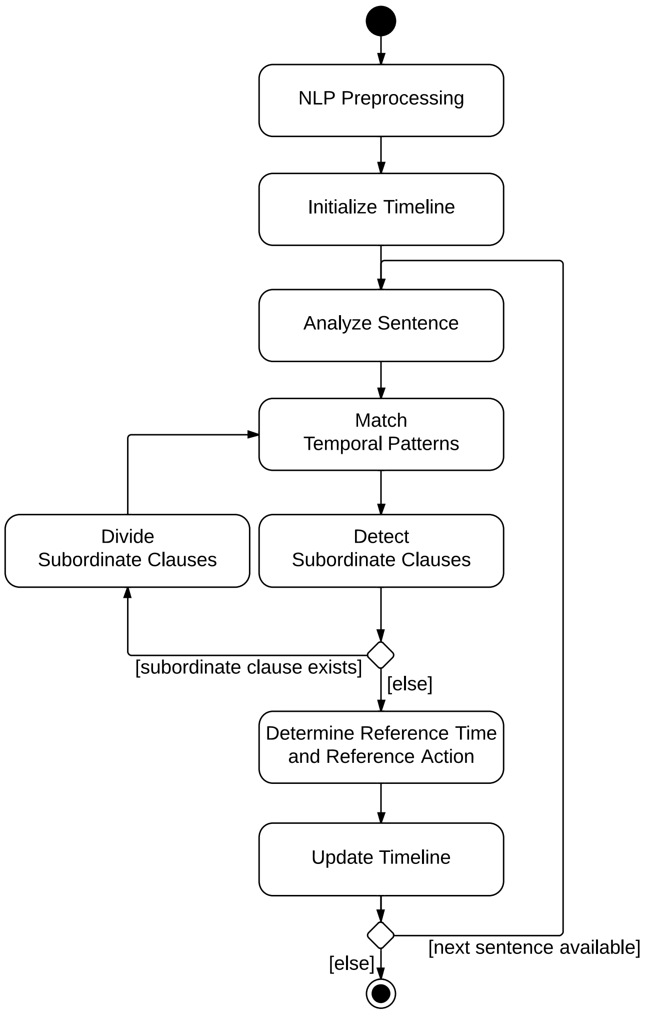

NLP Preprocessing
Initialize Timeline
Analyze Sentence
Match Temporal Patterns
Divide Subordinate Clauses
Detect Subordinate Clauses
[subordinate clause exists]
[else]
Determine Reference Time and Reference Action
Update Timeline
[next sentence available]
[else]

Figure 4: Schematic Overview of the Analysis Approach.

the preceding action, so the anchor action is the action *says (g)* in the previous sentence, resulting in $before(g, e)$. The second part *then glides away* contains the action *glides (f)*. Again, this means that the action is the transfer action of the TE. *Then* also references the preceding action; in this case the action *flaps* of the first part of the sentence. *Then* places the transfer action directly after the anchor action: $after(e, f)$.

In combination, both rearrangements produce the temporal order [e, f, g]. This order is already in our time line, so no changes occur.

After the last sentence is analyzed the resulting time line is the one shown in Figure 3b which enumerates the actions in the same order as the strictly sequential example.

5. EVALUATION

We evaluated our approach with 24 scripts for three different animations: Descriptions of the animation "Bunny" contain some TEs but they result in no rearrangements. Descriptions of the other two animations make heavy use of TEs and rearrangements because we directly asked the authors to do so. Some authors wrote descriptions of several animations but each author described every animation

Table 3: Evaluation Results for Three Different Animations.

Animation	Texts	TEs	✓	×	!	↔
Bunny	4	16	15	1	0	1
Cheerleader	10	81	67	5	9	6
Dragon	10	69	60	1	8	2
Total	24	166	142 86%	7 4%	17 10%	9

only once. Table 3 summarizes the results of our evaluation: It shows how many TEs there are in the texts (TEs), how many were correctly handled (✓), misinterpreted (×), or missed (!) and how many rearrangements (↔) were needed to correct the resulting time lines.

At first, we manually analyzed all descriptions for TEs and resulting rearrangements. Then we checked, whether the software correctly analyzed the texts. Some TEs do not result in rearrangements. In total, our software misinterpreted 4% of the TEs and missed 10%.

To evaluate the resulting time lines we compared them with the correct time lines. While analyzing the descriptions, we learned that some authors had used so many TEs that their texts do not describe the animation in the proper order. Some authors missed (or added) actions that were (not) in the animation. Therefore we manually created a correct time line for every description and then compared the computed time line with it.

A single misinterpreted (or missed) TE can result in many actions that are not at the correct position in the time line. Therefore, we cannot simply count the differences between the correct and the computed time lines: A time line $[a_0, a_1, a_2, a_3]$ with the actions a_i would have a difference count of four with respect to the time line $[a_3, a_0, a_1, a_2]$. Because of that, we decided to count the minimal rearrangements needed to repair the computed time line; this results in a count of one for the aforementioned example. As Table 3 shows, only nine of twenty-four mistakes (seven errors and 17 misses) surface in the resulting time line. This is because we start with the naïve time line and some TE do not change the order of the actions (such as *then* at the beginning of a sentence).

Three of the seven errors result from errors of the Stanford Parser. If the parser delivered correct output, no misinterpretations would have occurred. One further error is due to a formulation that was not allowed: One author wrote that an action happens *before and after* another action. Our approach is able to put the action either before or after the anchor action; it does not replicate an action because of TEs. The three remaining errors are misinterpretations of our software.

Three expressions were not detected because the respective formulations were not discovered during our analysis of the English language; these formulations were added to our software after the evaluation.

One missed expression indicates that an action should be inserted between two other actions: For example, *do a three times. After the second time, do b.* AliceNLP records the repetition of *a* as a single action on the time line. It does not break up the action.

The remaining thirteen missing expressions stem from one subject only. The subject wrote rather convoluted texts and often referenced actions in the description. Co-referencing is not fully implemented in AliceNLP. References that use the exact same wording can be resolved automatically but paraphrased actions cannot. Because many of these formulations used the expression *after* and referenced the directly preceding action, only few rearrangements had to be made. Thus, missing these expressions rarely results in an error in the final time line.

Even though the software did not produce the correct time line for every text, the results are promising: Overall, AliceNLP misses only 10% of the TEs and 142 of the 147 detected TEs are correctly interpreted. Many of the missed TEs do not lead to a rearrangement, so 17 of 24 time lines are correct. The incorrect time lines can be fixed with at most two reorderings. Some of the errors can be resolved by extending the covered expressions; some errors arise from incorrect input. Most errors can be remedied with a co-reference analysis for actions; if we provide these co-references manually, the software can analyze the TEs correctly. Automatic co-reference analysis for actions and handling simultaneous actions is future work.

6. CONCLUSION

Determining the correct order of actions in a narrative is essential for programming in natural language. A simple heuristics helps in reordering actions. The approach is not limited to programming in natural language: Automatically computed time lines can help in understanding error reports and support requests. Determining the sequence of actions leading to a failure is crucial to understanding the problem. Yet, users tend to provide seemingly important information first and fill in details as they cross their minds. Automatically derived time lines could also be used in question answering systems and big data analysis.

Even though our results are promising, much work remains to be done: Co-referencing actions in a natural language text is far from complete. Including a comprehensive co-reference analysis for actions into our software would drastically reduce the number of missed expressions. The list of covered TEs is likely to be expanded in the future.

Simultaneous actions can be detected using TEs as well: We will integrate patterns such as *when a, do b* and *while doing a, do b* into AliceNLP in future work. The operators for parallel patterns will fork (and join) the time line to model simultaneous actions.

7. REFERENCES

[1] B. W. Ballard and A. W. Biermann. Programming in natural language: NLC as a prototype. In *Proceedings of the 1979 annual conference*, ACM '79, pages 228–237, New York, NY, USA, 1979. ACM.

[2] M. J. Conway. *Alice: Easy-to-Learn 3D Scripting for Novices.* PhD thesis, Faculty of the School of Engineering and Applied Science, University of Virginia, Dec. 1997.

[3] R. Knöll and M. Mezini. Pegasus: first steps toward a naturalistic programming language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 542–559, New York, NY, USA, 2006. ACM.

[4] H. Lieberman and H. Liu. Feasibility studies for programming in natural language. In H. Lieberman,

F. Paternò, and V. Wulf, editors, *End User Development*, volume 9 of *Human-Computer Interaction Series*, pages 459–473. Springer Netherlands, 2006.

[5] H. Liu and H. Lieberman. Metafor: visualizing stories as code. In *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, pages 305–307, New York, NY, USA, 2005. ACM.

[6] R. Mihalcea, H. Liu, and H. Lieberman. Nlp (natural language processing) for nlp (natural language programming). In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 3878 of *Lecture Notes in Computer Science*, pages 319–330. Springer Berlin Heidelberg, 2006.

[7] G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[8] B. A. Myers. Natural programming: Project overview and proposal. Technical Report CMU-HCIL-98-100, Human-computer Interaction Institute, Carnegie Mellon University, Pittsburg, PA, USA, Jan. 1998.

[9] H. J. Ohlbach. Computational treatment of temporal notions: The cttn-system. In F. Schilder, G. Katz, and J. Pustejovsky, editors, *Annotating, Extracting and Reasoning about Time and Events*, volume 4795 of *Lecture Notes in Computer Science*, pages 72–87. Springer Berlin Heidelberg, 2007.

[10] J. F. Pane, B. A. Myers, and C. A. Ratanamahatana. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Hum.-Comput. Stud.*, 54(2):237–264, Feb. 2001.

[11] D. Price, E. Rilofff, J. Zachary, and B. Harvey. NaturalJava: a natural language interface for programming in Java. In *Proceedings of the 5th international conference on Intelligent user interfaces*, IUI '00, pages 207–211, New York, NY, USA, 2000. ACM.

[12] J. Pustejovsky, P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, and M. Lazo. The timebank corpus. In *Proc. of Corpus Linguistics*, pages 647–656, 2003.

[13] J. Pustejovsky, B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz, and I. Mani. The specification language timeml. *The language of time: A reader*, pages 545–557, 2005.

[14] J. Pustejovsky, R. Knippen, J. Littman, and R. Sauri. Temporal and event information in natural language text. *Language Resources and Evaluation*, 39(2-3):123–164, 2005.

[15] S. Russell and P. Norvig. *The Artificial Intelligence.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2010.

[16] S. K. Sanampudi and G. V. Kumari. Temporal reasoning in natural language processing: A survey. *International Journal of Computer Applications*, 1(4):53–57, 2010.

[17] F. Schilder. Event extraction and temporal reasoning in legal documents. In F. Schilder, G. Katz, and J. Pustejovsky, editors, *Annotating, Extracting and Reasoning about Time and Events*, volume 4795 of *Lecture Notes in Computer Science*, pages 59–71. Springer Berlin Heidelberg, 2007.

[18] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *Proceedings of the ACL 2013*, 2013.