Generating Animations from Screenplays

Generating Animations from Screenplays

Yeyao Zhang[1,3], **Eleftheria Tsipidi**[1] ,
Sasha Schriber[1], **Mubbasir Kapadia**[1,2], **Markus Gross**[1,3], **Ashutosh Modi**[1]
[1]Disney Research, [2]Rutgers University, [3]ETH Zurich
yezhang@inf.ethz.ch
{etsipidi,sasha.schriber}@disneyresearch.com
mubbasir.kapadia@rutgers.edu
{gross,ashutosh.modi}@disneyresearch.com

Abstract

Automatically generating animation from natural language text finds application in a number of areas e.g. movie script writing, instructional videos, and public safety. However, translating natural language text into animation is a challenging task. Existing text-to-animation systems can handle only very simple sentences, which limits their applications. In this paper, we develop a text-to-animation system which is capable of handling complex sentences. We achieve this by introducing a text simplification step into the process. Building on an existing animation generation system for screenwriting, we create a robust NLP pipeline to extract information from screenplays and map them to the system's knowledge base. We develop a set of linguistic transformation rules that simplify complex sentences. Information extracted from the simplified sentences is used to generate a rough storyboard and video depicting the text. Our sentence simplification module outperforms existing systems in terms of BLEU and SARI metrics.We further evaluated our system via a user study: 68 % participants believe that our system generates reasonable animation from input screenplays.

1. Introduction

Generating animation from texts can be useful in many contexts e.g. movie script writing (Ma and Kevitt, 2006; Liu and Leung, 2006; Hanser et al., 2010), instructional videos (Lu and Zhang, 2002), and public safety (Johansson et al., 2004). Text-to-animation systems can be particularly valuable for screenwriting by enabling faster iteration, prototyping and proof of concept for content creators.

In this paper, we propose a text-to-animation generation system. Given an input text describing a certain activity, the system generates a rough animation of the text. We are addressing a practical setting, where we do not have any annotated data for training a supervised end-to-end system. The aim is not to generate a polished, final animation, but a *pre-visualization* of the input text. The purpose of the system is not to replace writers and artists, but to make their work more efficient and less tedious. We are aiming for a system which is robust and could be deployed in a production environment.

Existing text-to-animation systems for screenwriting (§2) visualize stories by using a pipeline of Natural Language Processing (NLP) techniques for extracting information from texts and mapping them to appropriate action units in the animation engine. The NLP modules in these systems translate the input text into predefined intermediate action representations and the animation generation engine produces simple animation from these representations.

Although these systems can generate animation from carefully handcrafted simple sentences, translating real screenplays into coherent animation still remains a challenge. This can be attributed to the limitations of the NLP modules used with regard to handling complex sentences. In this paper, we try to address the limitations of the current text-to-animation systems. Main contributions of this paper are:

- We propose a screenplay parsing architecture which generalizes well on different screenplay formats (§3.1).
- We develop a rich set of linguistic rules to reduce complex sentences into simpler ones to facilitate information extraction (§3.2).
- We develop a new NLP pipeline to generate animation from actual screenplays (§3).

The potential applications of our contributions are not restricted to just animating screenplays. The techniques we develop are fairly general and can be used in other applications as well e.g. in-

*Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM)*, pages 292–307
Minneapolis, June 6–7, 2019.

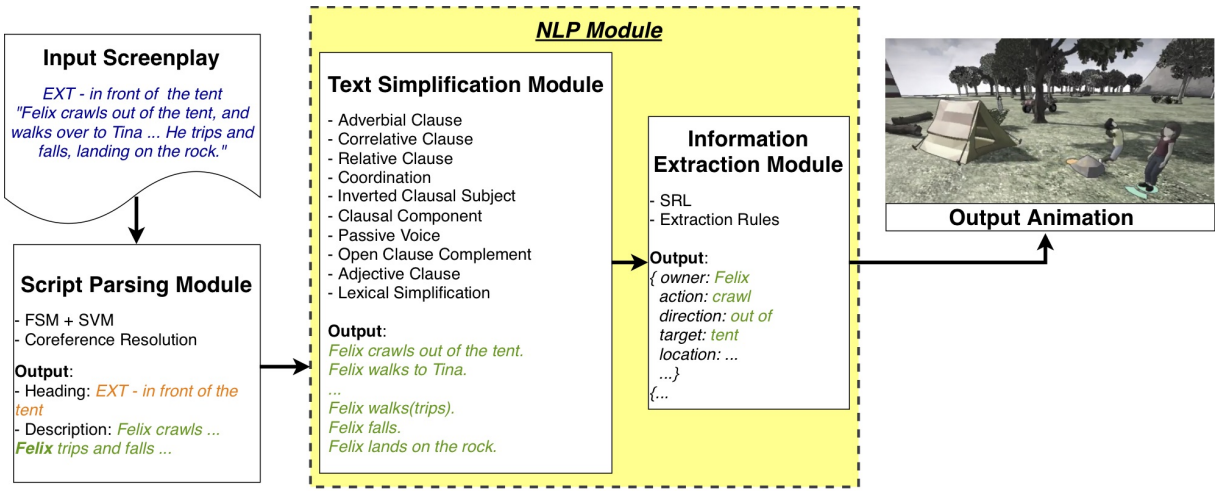


Figure 1: System Architecture: Screenplays are first segmented into different functional blocks. Then, the descriptive action sentences are simplified. Simplified sentences are used to generate animation.

formation extraction tasks.

2. Related Work

Translating texts into animation is not a trivial task, given that neither the input sentences nor the output animations have a fixed structure. Prior work addresses this problem from different perspectives (Hassani and Lee, 2016).

CONFUCIUS (Ma and Kevitt, 2006) is a system that converts natural language to animation using the FDG parser (Tapanainen and Järvinen, 1997) and WordNet (Miller, 1995). ScriptViz (Liu and Leung, 2006) is another similar system, created for screenwriting. It uses the Apple Pie parser (Sekine, 1998) to parse input text and then recognizes objects via an object-specific reasoner. It is limited to sentences having conjunction between two verbs. SceneMaker (Hanser et al., 2010) adopts the same NLP techniques as proposed in CONFUCIUS (Ma and Kevitt, 2006) followed by a context reasoning module. Similar to previously proposed systems, we also use dependency parsing followed by linguistic reduction (§3.2).

Recent advances in deep learning have pushed the state of the art results on different NLP tasks (Honnibal and Johnson, 2015; Wolf et al., 2018; He et al., 2017). We use pre-trained models for dependency parsing, coreference resolution and SRL to build a complete NLP pipeline to create intermediate action representations. For the action representation (§3.4), we use a key-value pair structure inspired by the PAR architecture (Badler et al., 2000), which is a knowledge base of representations for actions performed by virtual agents.

Our work comes close to the work done in the area of Open Information Extraction (IE) (Niklaus et al., 2018). In particular, to extract information, Clause-Based Open IE systems (Del Corro and Gemulla, 2013; Angeli et al., 2015; Schmidek and Barbosa, 2014) reduce a complex sentence into simpler sentences using linguistic patterns. However, the techniques developed for these systems do not generalize well to screenplay texts, as these systems have been developed using well-formed and factual texts like Wikipedia, Penn TreeBank, etc. An initial investigation with the popular Open IE system OLLIE (Open Language Learning for Information Extraction) (Mausam et al., 2012) did not yield good results on our corpus.

Previous work related to information extraction for narrative technologies includes the CARDINAL system (Marti et al., 2018; Sanghrajka et al., 2018), as well as the conversational agent PICA (Falk et al., 2018). They focus on querying knowledge from stories. The CARDINAL system also generates animations from input texts. However, neither of the tools can handle complex sentences. We build on the CARDINAL system. We develop a new NLP module to support complex sentences and leverage the animation engine of CARDINAL.

Recently, a number of end-to-end image generation systems have been proposed (Mansimov et al., 2015; Reed et al., 2016). But these systems do not synthesize satisfactory images yet, and are not suitable for our application. It is hoped that the techniques proposed in this paper could be used for automatically generating labelled da-

ta (e.g. (text,video) pairs) for training end-to-end text-to-animation systems.

3. Text-to-Animation System

We adopt a modular approach for generating animations from screenplays. The general overview of our approach is presented in Figure 1. The system is divided into three modules:

- **Script Parsing Module:** Given an input screenplay text, this module automatically extracts the relevant text for generating the animation (§3.1).
- **NLP Module:** It processes the extracted text to get relevant information. This has two submodules:
  - **Text Simplification Module:** It simplifies complex sentences using a set of linguistic rules (§3.2).
  - **Information Extraction Module:** It extracts information from the simplified sentences into pre-defined action representations (§3.4).
- **Animation Generation Module:** It generates animation based on action representations (§3.5).

3.1. Script Parsing Module

Typically, *screenplays* or *movie scripts* or *scripts* (we use the terms interchangeably), are made of several scenes, each of which corresponds to a series of consecutive motion shots. Each scene contains several functional components[1]: *Headings* (time and location), *Descriptions* (scene description, character actions), *Character Cues* (character name before dialog), *Dialogs* (conversation content), *Slug Lines* (actions inserted into continuous dialog) and *Transitions* (camera movement). In many scripts, these components are easily identifiable by indentation, capitalization and keywords. We call these scripts *well-formatted*, and the remaining ones *ill-formatted*. We want to segment the screenplays into components and are mainly interested in the Descriptions component for animation generation.

Well-formatted Scripts: We initially tried ScreenPy (Winer and Young, 2017) to annotate the well-formatted scripts with the component labels. However, ScreenPy did not perform well on our data. We developed our own model, based on Finite State Machines (FSM), for parsing scripts (for details refer to Appendix A). Due to space limitations, we do not describe the FSM model; the key idea is that the model uses hand-crafted transition rules to segment the input screenplay and generates (paragraph, component name) pairs.

Algorithm 1 Syntactic Simplification Procedure

```
procedure SYNTACTIC SIMPLIFICATION(sent, temp)
    Q ← empty queue
    HS ← empty integer hash set
    RES ← empty list
    Q.push(sent)                                  ▷ push input sentence to queue
    while Q ≠ Empty do
        str ← Q.pop()
        if hash(str) ∈ HS then
            RES.append(str)                       ▷ have seen this sentence
            continue
        HS.add(str)                               ▷ mark current sentence as already seen
        transform ← False
        for a in analyzers do
            if !transform & a.identify(str) then
                transform ← True
                simplified = a.transform(str)
                correct_verb_tense(simplified)
                Q.push(simplified)
        if transform ≠ True then
            RES.append(str)
    return RES
```

Ill-formatted Scripts: Taking all the (paragraph, component name) pairs generated by the FSM as ground truth, an SVM model is trained to segment ill-formatted screenplays with inconsistent indentations. For extracting features, each paragraph is encoded into a fix-sized vector using a pre-trained Universal Sentence Encoder. The SVM is trained and tested on a 9:1 train-test data split. The result shows an accuracy of 92.72 % on the test set, which is good for our purposes, as we are interested mainly in the Descriptions component.

Coreference Resolution: Screenplays contain a number of ambiguous entity mentions (e.g. pronouns). In order to link mentions of an entity, an accurate coreference resolution system is required. The extracted Descriptions components are processed with the NeuralCoref[2] system. Given the text, it resolves mentions (typically pronouns) to the entity they refer to in the text. To facilitate entity resolution, we prepend each Description component with the Character Cues component which appears before it in the screenplay (e.g. [character]MARTHA: [dialog]"I knew it!" [description]She then jumps triumphantly → MARTHA. She then jumps triumphantly).

3.2. Text Simplification Module

In a typical text-to-animation system, one of the main tasks is to process the input text to extract

[1] https://www.storysense.com/format.htm

[2] github.com/huggingface/neuralcoref

Syntactic Structure	*Identify* procedure	*Transform* procedure
Coordination	search if *cc* and *conj* in dependency tree	cut *cc* and *conj* link. If *conj* is verb, mark it as new root; else replace it with its sibling node.
Pre-Correlative Conjugation	locates position of keywords: "either", "both","neither"	removed the located word from dependency tree
Appositive Clause	find *appos* token and its head (none)	glue appositive noun phrase with "to be"
Relative Clause	find *relcl* token and its head	cut *appos* link, then traverse from root. Then, if no "wh" word present, put head part after anchor part; else, we divide them into 5 subcases (Table 2)
Adverbial Clause Modifier	find *advcl* token and its head. Also conjuncts of head token	cut *advcl* edge. If *advcl* token does not have subject, add subject of root as *advcl*'s most-left child and remove *prep* and *mark* token. Then traverse from both root and *advcl* token
Inverted Clausal Subject	*attr* token has to be the child of head of *csubj* token	change position of actual verb and subject
Clausal Complement	find *ccomp* token in dependency tree	cut *ccomp* link, add subject to subordinate clause if necessary
Passive Voice	check presence of *nsubjpass* or *csubjpass* optionally for *auxpass* and *agent*	cut *auxpass* link if any. Cut *nsubjpass* or *csubjpass* link. Prepend subject token to verb token's right children. Finally append suitable subject.
Open Clause Complement	find *xcomp* verb token adn actual verb token	if *aux* token presents, cut *aux* link, then replace xcomp-verb in subject's children with actual-verb, traverse from actual-verb; else, cut *xcomp* link, traverse from xcomp-verb
Adjective Clause	find *acl* verb token and its head	cut *acl* link. Link subject node with it. Traverser from *acl* node

Table 1: Linguistic rules for text simplification module

the relevant information about actions (typically verbs) and participants (typically subject/object of the verb), which is subsequently used for generating animation. This works well for simple sentences having a single verb with one subject and one (optional) object. However, the sentences in a screenplay are complicated and sometimes informal. In this work, a sentence is said to be complicated if it deviates from easily extractable and simple subject-verb-object (and its permutations) syntactic structures and possibly has multiple actions mentioned within the same sentence with syntactic interactions between them. By syntactic structure we refer to the dependency graph of the sentence.

In the case of screenplays, the challenge is to process such complicated texts. We take the text simplification approach, i.e. the system first simplifies a complicated sentence and then extracts the relevant information. Simplification reduces a complicated sentence into multiple simpler sentences, each having a single action along with its participants, making it straightforward to extract necessary information.

Recently, end-to-end Neural Text Simplification (NTS) systems (Nisioi et al., 2017; Saggion, 2017) have shown reasonable accuracy. However, these systems have been trained on factual data such as Wikipedia and do not generalize well to screenplay texts. Our experiments with such a pretrained neural text simplification system did not yield good results (§5.1). Moreover, in the context of text-to-animation, there is no standard labeled corpus to train an end-to-end system.

There has been work on text simplification using linguistic rules-based approaches. For example, (Siddharthan, 2011) propose a set of rules to manipulate sentence structure to output simplified sentences using syntactic dependency parsing. Similarly, the YATS system (Ferrés et al., 2016) implements a set of rules in the JAPE language (Cunningham et al., 2000) to address six syntactic structures: **Passive Constructions, Appositive Phrases, Relative Clauses, Coordinated Clauses, Correlated Clauses** and **Adverbial Clauses**. Most of the rules focus on case and tense correction, with only 1-2 rules for sentence splitting. We take inspiration from the YATS system, and our system incorporates modules to identify and transform sentence structures into simpler ones using a broader set of rules.

In our system, each syntactic structure is handled by an *Analyzer*, which contains two processes: *Identify* and *Transform*. The *Identify* process takes in a sentence and determines if it contains a particular syntactic structure. Subsequently, the *Transform* process focuses on the first occurrence of the identified syntactic structure and then splits and assembles the sentence into simpler sentences. Both *Identify* and *Transform* use Part-of-Speech (POS) tagging and dependency parsing (Honnibal and Montani, 2017) modules implemented in spaCy 2.0[3]

The simplification algorithm (Algorithm 1) starts with an input sentence and recursively processes it until no further simplification is possible. It uses a queue to manage intermediate simplified sentences, and runs in a loop until the queue is empty. For each sentence, the system applies each syntactic analyzer to *Identify* the correspon-

[3] https://spacy.io

Type	Example Input Sentence	System Output Sentence 1	System Output Sentence 2
Coordination	She LAUGHS, and[**cc**] gives[**conj**] Kevin a kiss.	She laughs.	She gives Kevin a kiss.
Pre-Correlative	It's followed by another squad car, both[**preconj**] with sirens blaring.	It's followed by another squad car, with sirens blaring.	–
Appositive	Kevin is reading a book the Bible[**appos**]	Kevin reads a book	The book is the Bible.
Relative-dobj	She pulls out a letter *which*[**dobj**] she hands[**relcl**] to Keven	Shee pulls out a letter	She hands a lettre to Kevin.
Relative-pobj	A reef encloses the cove *where*[**pobj**] he came[**relcl**] from.	A reef encloses the cove	he comes from the cove.
Relative-nsubj	Frank gestures to the SALESMAN, *who*[**nsubj**]'s waiting[**relcl**] on a woman	the SALESMAN waits on a woman.	Frank gestures to the SALESMAN.
Relative-advmod	Chuck is in the stage of exposure *where*[**advmod**] the personality splits[**relcl**]	Chuck is in the stage of exposure	the personality splits at exposure.
Relative-poss	The girl, *whose*[**poss**] name is[**relcl**] Helga, cowers.	The girl cowers	The girl 's name is Helga
Relative-omit	Kim is the sexpot Peter saw[**relcl**] in Washington Square Park	Peter sees Kim in Washington Square Park.	Kim is the sexpot.
Adverbial	Jim panics as[**advcl**] his mom reacts, shocked.	Jim panics, shocked.	Jim's mom reacts.
Adverbial-remove	Suddenly there's a KNOCK at the door, immediately after[**prep**] which JIM'S MOM enters[**advcl**].	Suddenly there 's a KNOCK at the door.	Immediately JIM 'S MOM enters.
Inverted Cl. Subject	Running[**csubj**] towards Oz is Steve Stifler	Steve Stifler runs towards Oz.	–
Clausal Component	The thing is, it actually sounds[**ccomp**] really good.	The thing is.(will be eliminated by the filter)	It actually sounds really good.
Passive Voice	They[**nsubjpass**] are suddenly illuminated by the glare of headlights.	Suddenly the glare of headlights illuminateds them.	–
Open Clausal	The sophomore comes running[**xcomp**] through the kitchen.	The sophomore runs through the kitchen.	The sophomore comes.
Adjective	Stifler has a toothbrush hanging[**acl**] from his mouth.	A toothbrush hangs from Stifler's mouth.	Stifler has a toothbrush.

Table 2: Syntactic simplification rules applied on example sentences.

ding syntactic structure in the sentence (line 14). If the result is positive, the sentence is processed by the *Transform* function to convert it to simple sentences (line 16). Each of the output sentences is pushed by the controller into the queue (line 19). The process is repeated with each of the *Identify* analyzers (line 13). If none of the analyzers can be applied, the sentence is assumed to be simple and it is pushed into the result list (line 21). We summarize linguistic rules in Table 1 and examples are given in Table 2. Next, we describe the coordination linguistic rules. For details regarding other rules, please refer to Appendix B.

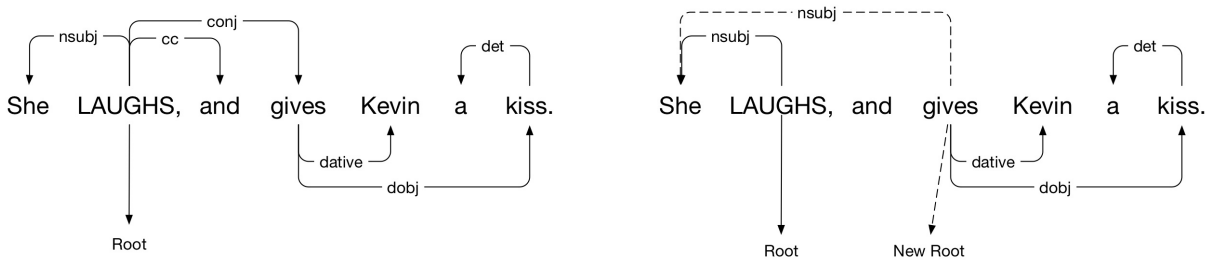
Coordination: Coordination is used for entities having the same syntactic relation with the head and serving the same functional role (e.g. subj, obj, etc.). It is the most important component in our simplification system. The parser tags word units such as "and" and "as well as" with the dependency label *cc*, and the conjugated words as *conj*. Our system deals with coordination based on the dependency tag of the conjugated word.

In the case of coordination, the *Identify* function simply returns whether *cc* or *conj* is in the dependency graph of the input sentence. The *Transform* function manipulates the graph structure based on the dependency tags of the conjugated words as shown in Figure 2. If the conjugated word is a verb, then we mark it as another root of the sentence. Cutting *cc* and *conj* edges in the graph and traversing from this new root results in a new sentence parallel to the original one. In other cases, such as the conjugation between nouns, we simply replace the noun phrases with their siblings and traverse from root again.

3.3. Lexical Simplification

In order to generate animation, actions and participants extracted from simplified sentences are mapped to existing actions and objects in the animation engine. Due to practical reasons, it is not possible to create a library of animations for all possible actions in the world. We limit our library to a predefined list of 52 actions/animations, expanded to 92 by a dictionary of synonyms (§3.5). We also have a small library of pre-uploaded objects (such as "campfire", "truck" and others).

To animate unseen actions not in our list, we use a word2vec-based *similarity* function to find the nearest action in the list. Moreover, we use WordNet (Miller, 1995) to exclude antonyms. This helps to map non-list actions (such as "squint at") to the similar action in the list (e.g. "look"). If we fail to find a match, we check for a mapping while including the verb's preposition or syntactic object. We also use WordNet to obtain hypernyms for further checks, when the *similarity* function fails to find a close-enough animation. Correspondingly, for objects, we use the same *similarity* function and WordNet's holonyms.



(a) original dependency graph

(b) dependency graph after *Transform*

Figure 2: *Transform* in an example coordination sentence. Firstly the dependency links of *cc* and *conj* are cut. Then we look for a noun in the left direct children of the original root *LAUGHS* and link the new root *gives* with it. In-order traverse from the original root and the new root will result in simplified sentences as shown in Table 2.

Our list of actions and objects is not exhaustive. Currently, we do not cover actions which may not be visual. For out of list actions, we give the user a warning that the action cannot be animated. Nevertheless, this is a work in progress and we are working on including more animations for actions and objects in our knowledge base.

3.4. Action Representation Field (ARF): Information Extraction

For each of the simplified sentences, information is extracted and populated into a predefined key-value pair structure. We will refer to the keys of this structure as *Action Representation Fields (ARFs)*. These are similar to entities and relations in Knowledge Bases. ARFs include: *owner*, *target*, *prop*, *action*, *origin_action*, *manner*, *modifier_location*, *modifier_direction*, *start-time*, *duration*, *speed*, *translation*, *rotation*, *emotion*, *partial_start_time* (for more details see Appendix C). This structure is inspired by the PAR (Badler et al., 2000) architecture, but adapted to our needs.

To extract the ARFs from the simplified sentences, we use a Semantic Role Labelling (SRL) model in combination with some heuristics, for example creating word lists for *duration, speed, translation, rotation, emotion*. We use a pre-trained Semantic Role Labelling model[4] based on a Bi-directional LSTM network (He et al., 2017) with pre-trained ELMo embeddings (Peters et al., 2018). We map information from each sentence to the knowledge base of animations and objects.

3.5. Animation Generation

We use the animation pipeline of the CARDINAL system. We plug in our NLP module in CARDINAL to generate animation. CARDINAL creates pre-visualizations of the text, both in storyboard form and animation. A storyboard is a series of pictures that demonstrates the sequence of scenes from a script. The animation is a 3-D animated video that approximately depicts the script. CARDINAL uses the Unreal game engine (Games, 2007) for generating pre-visualizations. It has a knowledge base of pre-baked animations (52 animations, plus a dictionary of synonyms, resulting in 92) and pre-uploaded objects (e.g. "campfire", "tent"). It also has 3-D models which can be used to create characters.

4. Text-to-Animation Corpus

We initially used a corpus of Descriptions components from ScreenPy (Winer and Young, 2017), in order to study linguistic patterns in the movie script domain. Specifically, we used the "heading" and "transition" fields from ScreenPy's published JSON output on 1068 movie scripts scraped from IMSDb. We also scraped screenplays from SimplyScripts and ScriptORama[5]. After separating screenplays into well-formatted and ill-formatted, Descriptions components were extracted using our model (§3.1). This gave a corpus of Descriptions blocks from 996 screenplays.

The corpus contains a total of 525,708 Descriptions components. The Descriptions components contain a total of 1,402,864 sentences. Out of all the Descriptions components, 49.45 % (259,973) contain at least one verb which is in the animation list (henceforth called "action verbs"). Descriptions components having at least one action verb have in total 920,817 sentences. Out of the-

[4] AllenNLP SRL model: https://github.com/allenai/allennlp

[5] http://www.simplyscripts.com and http://www.script-o-rama.com

Carl touches Ellie's shoulder as the doctor explains. Ellie drops her head in her hands.			
System output	Annotator I	Annotator II	$BLEU_2(\%)$
Carl touches Ellie 's shoulder	carl touches ellie's shoulder	carl touches ellie's shoulder.	38.73
the doctor explains	the doctor explains	the doctor is talking.	100
Ellie drops Ellie head in Ellie hands	ellie drops her head in her hands	ellie drops her head in her hands.	48.79

Table 3: Differences between system output and annotator responses

	$BLEU$	$SARI$
NTS-w2v	61.45	36.04
YATS	58.83	48.75
Our System	**67.68**	**50.65**

Table 4: Results on syntactic simplification

se, 42.2 % (388,597) of the sentences contain action verbs. In the corpus, the average length of a sentence is 12 words.

5. Evaluation and Analysis

There are no standard corpora for text-to-animation generation. It is also not clear how should such systems be evaluated and what should be the most appropriate evaluation metric. Nevertheless, it is important to assess how our system is performing. We evaluate our system using two types of evaluation: Intrinsic and Extrinsic. Intrinsic evaluation is for evaluating the NLP pipeline of our system using the BLEU metric. Extrinsic evaluation is an end-to-end qualitative evaluation of our text-to-animation generation system, done via a user study.

5.1. Intrinsic Evaluation

To evaluate the performance of our proposed NLP pipeline, 500 Descriptions components from the test set were randomly selected. Three annotators manually translated these 500 Descriptions components into simplified sentences and extracted all the necessary ARFs from the simplified sentences. This is a time intensive process and took around two months. 30 % of the Descriptions blocks contain verbs not in the list of 92 animation verbs. There are approximately 1000 sentences in the test set, with average length of 12 words. Each Descriptions component is also annotated by the three annotators for the ARFs.

Taking inspiration from the text simplification community (Nisioi et al., 2017; Saggion, 2017), we use the BLEU score (Papineni et al., 2002) for evaluating our simplification and information extraction modules. For each simplified sentence s_i we have 3 corresponding references r_i^1, r_i^2 and r_i^3. We also evaluate using the SARI (Xu et al., 2016) score to evaluate our text simplification module.

5.1.1. Sentence Simplification

Each action block a is reduced to a set of simple sentences $S_a = \{s_1, s_2, ...s_{n_a}\}$. And for the same action block a, each annotator t, $t \in \{1, 2, 3\}$ produces a set of simplified sentences $R_a^t = \{r_1^t, r_2^t, ...r_{m_a^t}^t\}$. Since the simplification rules in our system may not maintain the original ordering of verbs, we do not have sentence level alignment between elements in S_a and R_a^t. For example, action block a = *He laughs after he jumps into the water* is reduced by our system into two simplified sentences $S_a = \{s_1 = \textit{He jumps into the water}, s_2 = \textit{He laughs}\}$ by the temporal heuristics, while $annotator_3$ gives us $R_a^3 = \{r_1^3 = \textit{He laughs}, r_2^3 = \textit{He jumps into the water}\}$. In such cases, sequentially matching s_i to r_j will result in a wrong (hypothesis, reference) alignment which is (s_1, r_1^3) and (s_2, r_2^3).

To address this problem, for each hypothesis $s_i \in S_a$, we take the corresponding reference $r_i^t \in R_a^t$ as the one with the least Levenshtein Distance (Navarro, 2001) to s_i, i.e,

$$r_i^t = \arg\min_{r_j^t} lev\_dist(s_i, r_j^t), \forall j \in \{1, ..., m_a^t\}$$

As per this alignment, in the above example, we will have correct alignments (s_1, r_2^3) and (s_2, r_1^3). Thus, for each simplified sentence s_i we have 3 corresponding references r_i^1, r_i^2 and r_i^3. The aligned sentences are used to calculate corpus level BLEU score[6] and SARI score[7].

The evaluation results for text simplification are summarized in Table 4. We compare against YATS (Ferrés et al., 2016) and neural end-to-end text simplification system NTS-w2v (Nisioi et al.,

[6]We used NLTK's API with default parameters: http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.corpus_bleu

[7]Implementation available at https://github.com/cocoxu/simplification/

Field	$BLEU_1$	Field	$BLEU_1$
owner	56.16	**org_action**	80.92
target	41.85	**manner**	84.84
prop	28.98	**location**	71.89
action	70.46	**direction**	70.83
emotion	57.89		

Table 5: Results on textual ARFs (%)

Field	*P*	*R*	*F1*
s_time	86.49	68.63	76.53
rot.	82.04	81.16	81.60
duration	94.72	73.92	83.04
transl.	75.49	86.47	80.61
speed	94.41	79.50	86.32

Table 6: Result on Non-textual ARFs(%)

2017). YATS is also a rule-based text simplification system. As shown in Table 4, our system performs better than YATS on both the metrics, indicative of the limitations of the YATS system. A manual examination of the results also showed the same trend. However, the key point to note is that we are not aiming for text simplification in the conventional sense. Existing text simplification systems tend to summarize text and discard some of the information. Our aim is to break a complex sentence into simpler ones while preserving the information.

An example of a Descriptions component with $BLEU_2$ scores is given in Table 3. In the first simplified sentence, the space between *Ellie* and *'s* causes the drop in the score. But it gives exactly the same answer as both annotators. In the second sentence, the system output is the same as the annotator I's answer, so the $BLEU_2$ score is 1. In the last case, the score is low, as annotators possibly failed to replace *her* with the actual Character Cue *Ellie*. Qualitative examination reveals, in general, that our system gives a reasonable result for the syntactic simplification module. As exemplified, BLEU is not the perfect metric to evaluate our system, and therefore in the future we plan to explore other metrics.

5.1.2. ARF Evaluation

We also evaluate the system's output for action representation fields against gold annotations. In our case, some of the fields can have multiple (2 or 3) words such as *owner, target, prop, action, origin_action, manner, location* and *direction*. We use $BLEU_1$ as the evaluation metric to measure the BOW similarity between system output and ground truth references. The results are shown in Table 5.

In identifying *owner, target* and *prop*, the system tends to use a fixed long mention, while annotators prefer short mentions for the same character/object. The score of *prop* is relatively lower than all other fields, which is caused by a systematic SRL mapping error. The relatively high accuracy on the *action* field indicates the consistency between system output and annotator answers.

Annotation on the *emotion* ARF is rather subjective. Responses on the this field are biased and noisy. The $BLEU_1$ score on this is relatively low. For the other non-textual ARFs, we use precision and recall to measure the system's behavior. Results are shown in Table 6. These fields are subjective: annotators tend to give different responses for the same input sentence.

rotation and *translation* have Boolean values. Annotators agree on these two fields in most of the sentences. The system, on the other hand, fails to identify actions involving *rotation*. For example, in the sentence *"Carl closes CARL 's door sharply"* all four annotators think that this sentence involves rotation, which is not found by the system. This is due to the specificity of rules on identifying these two fields.

speed, *duration* and *start_time* have high precision and low recall. This indicates the inconsistency in annotators' answers. For example, in the sentence *"Woody runs around to the back of the pizza truck"*, two annotators give 2 seconds and another gives 1 second in *duration*. These fields are subjective and need the opinion of the script author or the director. In the future, we plan to involve script editors in the evaluation process.

5.2. Extrinsic Evaluation

We conducted a user study to evaluate the performance of the system qualitatively. The focus of the study was to evaluate (from the end user's perspective) the performance of the NLP component w.r.t. generating reasonable animations.

We developed a questionnaire consisting of 20 sentence-animation video pairs. The animations were generated by our system. The questionnaire was filled by 22 participants. On an average it took around 25 minutes for a user to complete the study.

We asked users to evaluate, on a five-point *Likert scale* (Likert, 1932), if the video shown was a reasonable animation for the text, how much of

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The animation shown in the video is a reasonable visualization of the text.	13.64 %	18.18 %	22.95 %	28.64 %	16.59 %
All the actions mentioned in the text are shown in the video.	15.68 %	20 %	22.96 %	20.68 %	20.68 %
All the actions shown in the video are mentioned in the text.	12.96 %	11.14 %	16.36 %	23.18 %	36.36 %

Table 7: User Study Results

the text information was depicted in the video and how much of the information in the video was present in the text (Table 7). The 68.18 % of the participants rated the overall pre-visualization as neutral or above. The rating was 64.32 % (neutral or above) for the conservation of textual information in the video, which is reasonable, given limitations of the system that are not related to the NLP component. For the last question, 75.90 % (neutral or above) agreed that the video did not have extra information. In general, there seemed to be reasonable consensus in the responses. Besides the limitations of our system, disagreement can be attributed to the ambiguity and subjectivity of the task.

We also asked the participants to describe qualitatively what textual information, if any, was missing from the videos. Most of the missing information was due to limitations of the overall system rather than the NLP component: facial expression information was not depicted because the character 3-D models are deliberately designed without faces, so that animators can draw on them. Information was also missing in the videos if it referred to objects or actions that do not have a close enough match in the object list or animations list. Furthermore, the animation component only supports animations referring to a character or object as a whole, not parts, (e.g. “Ben raises his head” is not supported).

However, there were some cases where the NLP component can be improved. For example, lexical simplification failed to map the verb “watches” to the similar animation “look”. In one case, syntactic simplification created only two simplified sentences for a verb which had three subjects in the original sentence. In a few cases, lexical simplification successfully mapped to the most similar animation (e.g.“argue” to “talk”) but the participants were not satisfied - they were expecting a more exact animation. We plan to address these shortcomings in future work.

6. Conclusion and Future Work

In this paper, we proposed a new text-to-animation system. The system uses linguistic text simplification techniques to map screenplay text to animation. Evaluating such systems is a challenge. Nevertheless, intrinsic and extrinsic evaluations show reasonable performance of the system. The proposed system is not perfect, for example, the current system does not take into account the discourse information that links the actions implied in the text, as currently the system only processes sentences independently. In the future, we would like to leverage discourse information by considering the sequence of actions which are described in the text (Modi and Titov, 2014; Modi, 2016). This would also help to resolve ambiguity in text with regard to actions (Modi et al., 2017; Modi, 2017). Moreover, our system can be used for generating training data which could be used for training an end-to-end neural system.

Acknowledgments

We would like to thank anonymous reviewers for their insightful comments. We would also like to thank Daniel Inversini, Isabel Simó Aynés, Carolina Ferrari, Roberto Comella and Max Grosse for their help and support in developing the Cardinal system. Mubbasir Kapadia has been funded in part by NSF IIS-1703883 and NSF S&AS-1723869.

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 344–354.

Norman I. Badler, Rama Bindiganavale, Jan Allbeck, William Schuler, Liwei Zhao, and Martha Palmer.

2000. Embodied conversational agents. chapter Parameterized Action Representation for Virtual Human Agents, pages 256–284. MIT Press, Cambridge, MA, USA.

Hamish Cunningham, Diana Maynard, and Valentin Tablan. 2000. Jape: a java annotation patterns engine.

Luciano Del Corro and Rainer Gemulla. 2013. Clausie: clause-based open information extraction. In *Proceedings of the 22nd international conference on World Wide Web*, pages 355–366. ACM.

Jessica Falk, Steven Poulakos, Mubbasir Kapadia, and Robert W Sumner. 2018. Pica: Proactive intelligent conversational agent for interactive narratives. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents*, pages 141–146. ACM.

Daniel Ferrés, Montserrat Marimon, Horacio Saggion, and Ahmed AbuRa'ed. 2016. Yats: Yet another text simplifier. In *Natural Language Processing and Information Systems*, pages 335–342, Cham. Springer International Publishing.

Epic Games. 2007. Unreal engine. *Online: https://www. unrealengine. com*.

Eva Hanser, Paul Mc Kevitt, Tom Lunney, and Joan Condell. 2010. Scenemaker: Intelligent multimodal visualization of natural language scripts. In *Proceedings of the 20th Irish Conference on Artificial Intelligence and Cognitive Science*, AICS'09, pages 144–153, Berlin, Heidelberg. Springer-Verlag.

Kaveh Hassani and Won-Sook Lee. 2016. Visualizing natural language descriptions: A survey. *ACM Comput. Surv.*, 49(1):17:1–17:34.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 473–483.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Richard Johansson, David Williams, Anders Berglund, and Pierre Nugues. 2004. Carsim: a system to visualize written road accident reports as animated 3d scenes. In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation*, pages 57–64. Association for Computational Linguistics.

R. Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Zhi-Qiang Liu and Ka-Ming Leung. 2006. Script visualization (scriptviz): a smart system that makes writing fun. *Soft Computing*, 10(1):34–40.

Ruqian Lu and Songmao Zhang. 2002. *Automatic generation of computeranimation: using AI for movie animation*. Springer-Verlag.

Minhua Ma and Paul Kevitt. 2006. Virtual human animation in natural language visualisation. *Artif. Intell. Rev.*, 25(1-2):37–53.

Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2015. Generating images from captions with attention. *arXiv preprint arXiv:1511.02793*.

Marcel Marti, Jodok Vieli, Wojciech Witoń, Rushit Sanghrajka, Daniel Inversini, Diana Wotruba, Isabel Simo, Sasha Schriber, Mubbasir Kapadia, and Markus Gross. 2018. Cardinal: Computer assisted authoring of movie scripts. In *23rd International Conference on Intelligent User Interfaces*, pages 509–519. ACM.

Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534. Association for Computational Linguistics.

George A. Miller. 1995. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38:39–41.

Ashutosh Modi. 2016. Event embeddings for semantic script modeling. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*.

Ashutosh Modi. 2017. *Modeling common sense knowledge via scripts*. Ph.D. thesis, Universität des Saarlandes.

Ashutosh Modi and Ivan Titov. 2014. Inducing neural models of script knowledge. In *Conference on Computational Natural Language Learning (CoNLL)*.

Ashutosh Modi, Ivan Titov, Vera Demberg, Asad Sayeed, and Manfred Pinkal. 2017. Modelling semantic expectation: Using script knowledge for referent prediction. *Transactions of the Association for Computational Linguistics*, 5:31–44.

Gonzalo Navarro. 2001. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88.

Thomas Wolf, James Ravenscroft, Julien Chaumond, and Maxwell Rebo. 2018. Neuralcoref: Coreference resolution in spacy with neural networks. Available online at https://github.com/huggingface/neuralcoref.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Appendix A

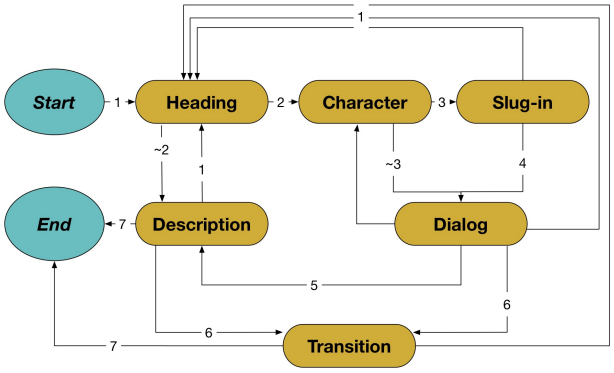


Figure 3: FSM of Well-formatted Screenplay Parser. Numbers of link are Rules and $\sim$ means logical NOT

ID	Rule Summary
1	If input is uppercase and contains heading words such as 'INT', 'EXT'. etc, return True, otherwise False
2	If input is uppercase and contains character words such as 'CONT.', '(O.S)', or if #indentation $>$ most frequent #indentation, return True. Otherwise False
3	If input starts with '(' , return True, otherwise False
4	If input ends with ')', return True, otherwise False
5	If $\lvert \#lastindents - \#currentindents \rvert < 3$, return True, otherwise False
6	If the input is uppercase and contains transition words such as 'DISSOLVE', 'CUT TO'. etc, return True, otherwise False
7	If the input equals to 'THE END', return True. Otherwise False.

Table 8: FSM Transition Rules

COMPLEX: Another parent , Mike Munson , sits on the bench with a tablet and uses an app to track and analyze the team 's shots.
NSELSTM-B: Another parent, Mike Munson, sits on the bench with a tablet and uses an app to track.
YATS: Another parent sits on the bench with a tablet and uses an app to track and examines the team' s shots. This parent is Mike Munson.
OURS: Another parent is Mike Munson. Another parent sits on the bench with a tablet. Another parent uses an app.
COMPLEX: Stowell believes that even documents about Lincoln's death will give people a better understanding of the man who was assassinated 150 years ago this April.
NSELSTM-B: Stowell believes that the discovery about Lincoln's death will give people a better understanding of the man.
YATS: Stowell believes that even documents about Lincoln' s death will give people a better reason of the man. This man was assassinated 150 years ago this April.
OURS: Stowell believes. Even documents about Lincoln 's death give people a better understanding of the man. Somebody assassinates the man 150 years ago this April.

Table 9: Example Model Outputs

Appendix B: Algorithms

Algorithm 2 Identify Adverbial Clause

```
procedure ADVERBIAL CLAUSE IDENTIFY PROCEDURE(sent)                ▷ The input sentence
    find tokens in sents with dependency tag ROOT and ADVCL (we call it advcl)
    if no ADVCL token in sents then
        return False
    find father token of advcl as father
    if father is not a VERB then
        we make it as a VERB                                       ▷ We correct POS error here
    find conjunction tokens of father as conjuncts
    if NOUN in advcl's left subtree then
        subject ← this NOUN
    else
        subject ← NOUN in father's left subtree
    return True
```

Algorithm 3 Transform Adverbial Clause

```
procedure ADVERBIAL CLAUSE TRANSFORM PROCEDURE(sent)               ▷ The input sentence
    cut edge between root and advcl token                          ▷ remove advcl token from root's children
    if advcl verb does not have its own subject then
        add subject as advcl's most left direct child
    remove PREP and MARK token in advcl's children, modify temporal id accordingly
    str1 ← traverse_a_string(root)
    str2 ← correct_tense(traverse_a_string(advcl))
    return [str1, str2]
```

Algorithm 4 Identify() in Relative Clause Analyzer

```
procedure RELATIVE CLAUSE IDENTIFY PROCEDURE(sent)                 ▷ The input sentence.
    if no RELCL token in sent then
        return False
    anchor ← RELCL token
    head ← anchor's head token
    wg ← NULL
    for token t in anchor's children do
        if t.tag ∈ [WDT, WP, WP$, WRB] then
            wh ← t
    return True
```

Algorithm 5 Transform() in Relative Clause Analyzer

```
procedure RELATIVE CLAUSE TRANSFORM PROCEDURE(root, anchor, head, wh)   ▷ Root of dependency tree, the recl anchor token, its head, and wh-word
    cut relcl edge between head and anchor
    str1 ← traverse_a_string(root)
    if wh = NULL then                                              ▷ No Wh-word in the sentence, concatenate
        str2 ←traverse_a_string(anchor) + traverse_a_string(head)
    else
        wh-dep ← dependency tag of wh
        wh-head ← head of wh
        remove wh in anchor's children
        if wh-dep= DOBJ then                                       ▷ wh is verb
            str2← traverse_a_string(wh-head) + traverse_a_string(anchor)
        else if wh-dep= POBJ then                                  ▷ wh-head is preposition
            put head after wh-head in anchor's children
            str2 ← traverse_a_string(anchor)
        else if wh-dep∈ [NSUBJ, NSUBJPASS] then                    ▷ wh is subject
            str2 ← traverse_a_string(wh-head) + traverse_a_string(anchor)
        else if wh-dep= ADVMOD then                                ▷ wh is time or location
            prep ← 'at'
            str2 ← traverse_a_string(anchor) + prep+ traverse_a_string(wh-head)
        else if wh-dep= POSS then                                  ▷ wh = whose
            str2 ← traverse_a_string(wh-head) + be verb+ traverse_a_string(anchor)
    correct verb tense in str1 and str2
    return [str1, str2]
```

Algorithm 6 Transform() in Coordination Analyzer

procedure COORDINATION TRANSFORM PROCEDURE($sents$) ▷ Input sentence
  results ← empty list
  find *root* of dependency tree of *sents*
  find first *cc*(if any) and *conj* token in denpendency tree and their head token *main*
  embed all conjugate words of *main* in a list *conjus*
  cut conj edge between *main* and *cc* and *conj*
  if no object for *main* **then**
    try find object in *conj*'s right children
  str1 ← traverse_a_string(*root*)
  results.append(str1)
  for *conj* in *conjs* **do**
    type ← get_conj_type(*main*, *conj*)
    if *type*=VERB&VERB **then**
      correct part-of-speech tag if necessary ▷ spaCy tends to tag verb as noun
      if *conj* has its own subject **then**
        new-root ← *conj*
      else
        if *main*=*root* **then**
          new-root ← *conj*
        else
          replace *main* with *conj* in *root*'s children
          new-root ← *root*
    else ▷ Other cases such as NOUN&NOUN, AD*&AD*, apply same rule
      main-head ← head of *main*
      replace *main* with *conj* in *main-head*'s children
      new-root ← *root*
    str2 ← traverse_a_string(*new-root*)
    results.append(str2)
  return results

Algorithm 7 Identify() in Passive Analyzer

procedure PASSIVE VOICE IDENTIFY PROCEDURE($sents$) ▷ Input sentence
  is-passive ← False
  for token *t* in sents **do**
    if t.dep ∈ [CSUBJPASS, NSUBJPASS] **then**
      subj-token ← t
      verb-token ← t.head
      is-passive ← True
    if t.dep ∈ [AUXPASS] and t.head = verb-token **then**
      auxpass-token ← t
    if t.dep ∈ [AGENT] and t.text = 'by' **then**
      by-token ← t
  return is-passive

Algorithm 8 Transform() in Passive Analyzer

procedure PASSIVE VOICE TRANSFORM PROCEDURE($sents$) ▷ Input sentence
  if auxpass-token ≠ NULL **then**
    cut auxpass edge
  cut nsubjpass or csubj edge
  prepend subject-token to verb-token's right children
  if by-token ≠ NULL **then**
    cut by-agent edge
    add by-token's right children to verb-token's left children
  else
    add 'Somebody' to verb-token's left children
  correct verb tense for verb-token
  return traverse_a_string(root-token)

Algorithm 9 Transform() in Appositive Clause Analyzer

procedure APPOSITIVE CLAUSE TRANSFORM PROCEDURE($anchor, head$) ▷ The APPOS token and its head token.
  cut edge between **anchor** and **head** token ▷ remove **anchor** from **head**'s right children
  str1 ← traverse_a_string(root token of input sentence)
  str2 ← traverse_a_string(**head**) + be_verb + traverse_a_string(**anchor**)
  correct verb tense in str1 and str2
  return [str1, str2]

Algorithm 10 Identify() in Inverted Clausal Subject Analyzer

procedure INVERTED CLAUSAL SUBJECT IDENTIFY PROCEDURE($sents$) ▷ Input sentence
  for Token t in *sents* **do**
    if t.dep = CSUBJ and t.tag ∈ [VBN, VBG] and t.head.lemma='be' **then**
      attr ← token with dependency label *attr* in t.head's right children
      if attr = NULL **then** ▷ *attr* is the actual subject of the sentence
        return False ▷ Make sure this is an inverted sentence
      actual-verb ← t
      be-verb ← t.head
      return True
  return False

Algorithm 11 Transform() in Inverted Clausal Subject Analyzer

procedure INVERTED CLAUSAL SUBJECT TRANSFORM PROCEDURE($sents$) ▷ Input sentence
  get access to actual-verb, be-verb and attr in identify procedure 10
  change position of actual-verb and attr in be-verb's children
  return [traverse_a_string(be-verb)]

Algorithm 12 Transform() in CCOMP Analyzer

procedure CLAUSE COMPONENT TRANSFORM PROCEDURE($sents$) ▷ Input sentence
  cut CCOMP link in dependency tree
  subject ← find ccomp verb's subject
  if subject ≠ NULL and subject ≠ DET(e.g. 'that') **then**
    if original verb do not have object **then**
      make subject as original verb's object
  else if subject = DET (e.g. 'that') **then**
    find root verb's object, substitute 'that'
  str1 ← traverse_a_string(ccomp verb)
  str2 ← traverse_a_string(original verb)
  return [str1, str2]

Algorithm 13 Transform() in XCOMP Analyzer

procedure OPEN CLAUSAL COMPONENT TRANSFORM PROCEDURE($sents$) ▷ Input sentence
  subject-token ← find_subject(xcomp-verb-token) ▷ find subject of the actual verb
  results ← empty list
  if AUX ∉ *sents* **then** ▷ for some cases two verbs needs to be output
    cut xcomp link
    results.add(traverse_a_string(xcomp-verb-token))
  remove AUX token in the dependency tree
  replace xcomp-verb-token in subject's children with actual-verb-token
  results.add(traverse_a_string(actual-verb-token))
  return results

Algorithm 14 Transform() in ACL Analyzer

procedure ADJECTIVE CLAUSE TRANSFORM PROCEDURE($sents$) ▷ Input Sentence
  cut acl edge in dependency tree
  str1 ← traverse_a_string(root-token)
  mid-fix ← empty string
  if acl-token.tag = VBN and by-token in acl-token's right children **then**
    mid-fix ← 'be'
  update acl-verb-token's left children with [acl-noun, mid-fix, [t ∈ acl-verb-token.lefts where t.dep ∉ [AUX]]
  correct acl-verb-tense
  str2 ← traverse_a_string(acl-verb-token)
  return [str1, str2]

Algorithm 15 Get-Temporal Function at Line 5 in Algorithm 3

procedure ADVERBIAL CLAUSE TEMPORAL INFO EXTRACTION PROCEDURE($prep-or-mark, cur-temp$)▷ The PREP or MARK token in input sentence
  flag ← False ▷ whether we change the temp
  if prep-or-mark.type = PREP **then**
    sign ← -1
  else
    sign ← 1
  text ← prep-or-mark.text.lower()
  if text = *as* **then**
    flag ← True
  else if text ∈ [*until*, *till*] **then**
    flag ← True
    cur-temp ← cur-temp + sign
  else if text = *after* **then**
    flag ← True
    cur-temp ← cur-temp - sign
  else if text = *before* **then**
    flag ← True
    cur-temp ← cur-temp + sign
  return [flag, cur-temp]

Appendix C: Action Representation Fields

Action Representation Fields (ARFs) in the demo sentence *James gently throws a red ball to Alice in the restaurant from back*, extracted with SRL:

- **owner**: James
- **target**: a red ball
- **prop**: to Alice
- **action**: throw
- **origin_action**: throws
- **manner**: gently
- **modifier_location**: in the restaurant
- **modifier_direction**: from back

In this case, our output for the *prop* and *target* is not correct; they should be swapped. This is one example where this module can introduce errors.

Additional ARFs, extracted heuristically:

- **startTime**: Calculated by current scene time
- **duration**: We have a pre-defined list of words that when appearing in the sentence, they will indicate a short duration (e.g "run" or "fast") and therefore the *duration* will be set to 1 second; in contrast, for words like "slowly" we assign a duration of 4 seconds; otherwise, the duration is 2 seconds.
- **speed**: Similarly to *duration*, we have pre-defined lists of words that would affect the speed of the pre-baked animation: "angrily" would result in faster movement, but "carefully" in slower movement. We have 3 scales: 0.5, 1, 2 which corresponds to *slow, normal* and *fast*.
- **translation**: We have a list of *actions* which would entail a movement in from one place to another, e.g. "go". If the value of the *action* exists in this list, it is set to True, otherwise False.
- **rotation**: If the *action* exists in our list of verbs that entail rotation, this field is True, otherwise False. Rotation refers to movement in place e.g. "turn" or "sit".
- **emotion**: We find the closet neighbor of each word in the sentence in list of words that indicate emotion, using word vector similarity. If the similarity exceeds an empirically tested threshold, then we take the corresponding emotion word as the *emotion* field of this action.
- **partial_start_time**: an important field, since it controls the sequence order of each action. It determines which actions happen in parallel and which happen sequentially. This is still an open question. We solve this problem when doing sentence simplification. Together with the input sentence, current time is also fed into each *Analyzer*. There are several rules in some of the *Analyzers* to obtain temporal information. For example, in Line 5 of the *Adverbial Clause Analyzer* (c.f.3), we assign different temporal sequences for simplified actions. The algorithm is shown in Algorithm 15. The *sign* together with specific prepositions determines the change direction of current temporal id. In the *Coordination Analyzer*, the current temporal id changes when it encounters two verbs sharing same subject. Then the later action will get a bigger temporal id.