Script Combination for Enhanced Story Understanding and Story Generation Systems

Megan McKenzie MMCKENZIE@SMITH.EDU
Alexis Kilayko AKILAYKO@SMITH.EDU
Jamie C. Macbeth JMACBETH@SMITH.EDU
Department of Computer Science, Smith College, Northampton, MA 01063 USA

Scott Carter SCOTT.CARTER@TRI.GLOBAL
Katharine Sieck KATE.SIECK@TRI.GLOBAL
Matthew Klenk MATT.KLENK@TRI.GLOBAL
Toyota Research Institute, 4440 El Camino Real, Los Altos, CA 94022, USA

Abstract

Scripts, knowledge structures defining sequences of events in stereotypical social situations, were traditionally used to simulate the ways in which people can infer unstated details in understanding a story. In this paper, we describe the MUltiple SCRipt AcTivator (MUSCRAT), and the Script Combination Applier Mechanism (SCAM), significant enhancements of Cullingford's Script Applier Mechanism which accomplish two novel aims. One system, MUSCRAT, is able to activate more than one script and use them during a story understanding process. The second system, SCAM, uses scripts for story generation, using script variables as "terminals" for combining two or more scripts together to create complex narrative situation structures. The combinational abilities of these systems are enhanced by the fact that the scripts are represented not using linguistic elements but using decompositions into abstract conceptual primitives. We also discuss how some of the perceived weaknesses of scripts stemming from prior work may be overcome in systems that represent general thinking and reasoning processes as combined instantiations of standardized and generalized memory episodes.

1. Introduction

A script is a form of knowledge that defines the stereotypical sequence of events in a common situation, allowing people to know what behavior is appropriate in that situation (Schank & Abelson, 1977). Scripts were traditionally used in story understanding systems to simulate the ways in which people can reference a memory structure about a common situation to recover unstated details in a story. Scripts have endured as a topic of interest in AI, cognitive science, psychology for more than 40 years. The rise of big data and machine learning enhanced NLP has inspired research in methods to collect scripts at large scale from large language corpora and other datasets (Chambers & Jurafsky, 2008, 2009; Rudinger et al., 2015; Boujarwah et al., 2012) toward the goal of building large-scale script repositories. Acknowledging that much of the common sense knowledge that needs to be embedded in scripts may not appear in large language corpora, other work exploits human-machine collaboration systems to collect more of the events that are typically elided in texts (Ciosici et al., 2021).

However, recent work on scripts is problematic because, in the process of using language corpora, it strives for a representation that is purely linguistic and strips away important traces of the original script representation—for example, role specifications for animate actors or inanimate objects. There has also been little advancement in understanding how scripts might interact with other higher-level knowledge structures, or how the idea of scripts could be generalized to subsume those higher-level structures (e.g., involving plans and goals).

This paper presents efforts to get "back to basics" with scripts, and the original ideas of scripts as, in the words of Schank & Abelson (1977), “an economy measure in the storage of episodes, when enough of them are alike they are remembered in terms of a standardized generalized episode”, with memory episodes represented as cognitive structures composed of language-free conceptual primitives. In this paper we present a pair of prototype script combination systems, one focused on understanding, the other focused on generation, to illustrate how standardized and generalized episode structures and their combinations could be broadly representative of thinking and reasoning processes.

One perceived weakness of scripts when used in understanding processes involves encountering unexpected events in a story which do not fit or match with the expected events based on the script. Allowing a system to combine scripts enriches the understanding process, since, if an understanding system has the ability to activate more than one script simultaneously, an event that appears unexpected in one script may be expected in another. As a result, the systems demonstrate how the perceived weaknesses of scripts stemming from prior work may be overcome in systems that represent general thinking and reasoning processes as instantiations of standardized and generalized memory episodes. The combinational abilities of these systems are enhanced by the fact that the scripts are represented not using linguistic elements but using decompositions into abstract conceptual primitives.

The paper is organized as follows. After reviewing general prior work, and the original Script Applier Mechanism (SAM), we present two novel enhancements of SAM that perform script combination. First we present, as a story understanding system, the MUltiple SCRipt AcTivator (MUSCRAT), which processes a story’s episodic events, and allows multiple scripts to be active and applied to the events in the episode. We then present a second system, the Script Combination Applier Mechanism (SCAM), which applies scripts outside of the traditional natural language story understanding context, using script variables as “terminals” for combining two or more scripts together to create complex narrative situation structures for story generation. Following a discussion of broader implications of script combination for reasoning and knowledge representation, the paper concludes with proposals of future work on the topic.

2. Background and Related Work

The advent of big data machine learning-enhanced NLP has inspired research in methods to collect scripts at large scale using "unsupervised" pattern recognition and large language corpora (Chambers & Jurafsky, 2008, 2009; Rudinger et al., 2015). Acknowledging that much of the common sense knowledge that should be embedded in scripts is typically elided and likely does not appear in large language corpora, later work has migrated towards collecting scripts through specialized curation systems using human-machine collaboration and crowd sourcing (Ciosici et al., 2021; Li et al., 2012; Boujarwah et al., 2012), viewing script extraction as language modeling (Rudinger et al., 2015). There have been attempts to use language models to capture "tracks" in scripts (Weber et al., 2018) and to capture stronger causal relations between events in scripts (Weber et al., 2020).

However, with some notable exceptions (Mueller, 2004) much of this work is evaluated through autogenerated narrative cloze tests which are known to be problematic (Chambers, 2017). More generally, researchers have struggled to mine language corpora for scripts while maintaining useful key elements of the original script events representations, such as actor and object roles, entry conditions, and causal relationships (Weber et al., 2020).

Schank and Abelson were originally inspired by Minsky's work on *frames* (Minsky, 1975) to invent the concept of a script in the context of building AI systems for natural language understanding and story understanding (Schank & Abelson, 1975, 1977). A script is a form of knowledge which defines the stereotypical sequence of events that occurs in a common situation and allows us to know what behavior is appropriate for a particular situation, both for ourselves and for others. Theoretically, the specific, detailed knowledge contained in a script helps us interpret instances of events that are new, but match patterns that we have previously and frequently experienced, allowing us to "fill in the gaps" in understanding a story (e.g., the classic "did Bob eat lobster?" for the $RESTAURANT script). At the most primitive level, Schank & Abelson describe scripts in terms of episodic memory as "As an economy measure in the storage of episodes, when enough of them are alike they are remembered in terms of a standardized generalized episode." Although Schank and Abelson invented the concept of a script in the context of building AI systems for natural language understanding and story understanding, the concept has received significant attention and frequent citation across many disciplines (e.g., "behavioral script theory") and is supported empirically (Bower et al., 1979).

In the same work that Schank and Abelson invented scripts, they also asserted that scripts alone, as they initially conceived them, were insufficient for human-like story understanding, and they proposed links between scripts and separate plan and goal structures. Scripts were perceived as being constraining and limited. According to some, scripts were intended to capture "only very limited stereotypic knowledge" (Dyer, 1982), or "knowledge about highly stereotypic situations" that is "tightly bound to well-specified situations," (Lehnert, 1977) and only a single script structure could be applied at one time when understanding a particular story. In natural language story understanding contexts, it was believed that scripts did not account for the goals and plans of story characters (Dyer, 1982), and the intentions and affective states of story characters. It was thought that scripts are only capable of working in a "top-down" manner, only fulfilling prior expectations while selectively ignoring unusual or unexpected events.

Much of subsequent work focused on how the perceived rigidity of scripts could, at least partially, be overcome by making them dynamic and allowing them to evolve with experience (Schank, 1983; Schank & Burstein, 1982). Work on dynamic memory theory and related understanding systems (Lebowitz, 1983; Kolodner, 1981) largely focused on learning and generalization over script structures that had strong similarities, for example, constructing a memory organization packet

(MOP) to represent the generalized concept of an office visit based on similarities and differences among experiences with visiting a doctor's office and experiences with visiting a lawyer's office (Schank, 1983).

But much of the focus in this work was on how a single script knowledge structure can adapt and change, not on how differing structures could be synthesized together. Little work has explored how relatively dissimilar scripts might be combined with one another as part of a process of understanding or as a general thinking process. Cullingford's original Script Applier Mechanism (Cullingford, 1977; Schank & Riesbeck, 1982) did allow for multiple scripts within a hierarchy to be activated simultaneously, and for high level scripts to contain "subscripts" or more detailed scripts nested within them (for example $SUBWAY-RIDE as a subscript of a high-level $TRIP script), but did not allow for scripts that might not normally be part of the same hierarchy to be combined in varieties of ways at the same level of detail. Granger's ARTHUR story understanding system combined scripts in order to recognize and resolve contradictions that arise when understanding "intentionally misleading" stories (Granger, 1980). Although ARTHUR instantiates multiple scripts in its understanding process, it uses specialized goal and plan tree structures as intermediaries between scripts, rather than a generalized episodic memory structure combination process that we explore in this work.

Our work is strongly contrasted with much of the more recent text mining work on scripts because we focus on representing the events and acts in scripts as structures composed of non-linguistic conceptual primitives (Schank, 1972, 1975) instead of linguistic narrative chains in text corpora, corresponding with Schank and Abelson's original exposition and corresponding with the original script applier mechanism system. Although the ad-hoc nature of CD is controversial (see, e.g., Slade, 1987; Winograd, 1978), this choice is supported by recent evidence that these kinds of meaning representations are more in line with representations that humans employ in language understanding (Macbeth et al., 2017). The hallmark of primitive decomposition systems is that they allow for a rich substrate of matchings between conceptual structures and their constituents, which is exactly the kind of representations desired for combining scripts.

3. The Script Applier Mechanism

In this section we provide background on the Script Applier Mechanism system prior to our enhancements, as a review of its parts and processes and in order to preface as well as motivate our work. The Script Applier Mechanism program put forth by Cullingford, referred to as SAM, is a story understanding computer program that intends to simulate the human ability to "fill in the gaps" by drawing upon script knowledge (Cullingford, 1977). The program receives a story to be processed, identifies the script being referenced, and recognizes the parts of the script that have occurred in the story. It then consults the script to infer the events that took place despite no explicit mention. When SAM terminates, it outputs its understanding of the story, which includes the explicit events from the input as well as the implicit events inserted by the program.

A less comprehensive version of SAM exists called MicroSAM, whose purpose, like SAM, is to apply scripts in order to understand stories. However, it offers considerably less functionality in

$SHOPPING, Text Version	$SHOPPING, Conceptual Dependency Version

1. Someone goes to a store.
2. She picks up an object.
3. The store transfers possession of the object to her.
4. She transfers possession of some money to the store.
5. She leaves the store.
	`(ptrans (actor ?shopper) (object ?shopper) (to ?store))`
	`(ptrans (actor ?shopper) (object ?bought) (to ?shopper))`
	`(atrans (actor ?store) (object ?bought)`
	`        (from ?store) (to ?shopper))`
	`(atrans (actor ?shopper) (object (money))`
	`        (from ?shopper) (to ?store))`
	`(ptrans (actor ?shopper) (object ?shopper)`
	`        (from ?store) (to ?elsewhere))`

Figure 1. An example of a script in the MicroSAM corpus, $SHOPPING, associated with the keyword STORE, and a program implementation of $SHOPPING in Common Lisp.

comparison to the original. As MicroSAM was made available to us, we have built our research upon its codebase.

3.1 Script corpus

MicroSAM contains a corpus of scripts as its knowledge base. A *script* is structured as a list of patterns, where a *pattern* represents a simple, individual action that the script can be broken down into. Every script is associated with a *keyword*, or a symbol whose occurrence in a story signifies that the particular script must be taking place. We refer to a specific script by the form $SCRIPT. An example of a script in the MicroSAM corpus, $SHOPPING, is shown in Figure 1. This script consists of five patterns and describes the typical sequence of events that occurs when an individual visits a store and purchases an item. It is associated with the keyword STORE.

Rather than natural language, scripts are implemented in MicroSAM as *Conceptual Dependency* (CD) forms. A CD structure consists primarily of an *action primitive* (ACT) conveying the action of the sentence, the actor who performs the action, and the object upon which the ACT is performed. Certain ACTs also take the cases "from" and "to," indicating direction. A program implementation of $SHOPPING in Common Lisp is shown in Figure 1. Each pattern is structured as an expression containing an action primitive followed by the actor, object, and directive cases as needed. The PTRANS primitive represents the movement of an object (which may be the same as the actor) from one location to another. The ATRANS primitive represents the transfer of possession or ownership of an object from one entity to another.

Variables, or roles in the script that are filled by different values for each story, are represented by the form ?variable. Here, the variables are ?shopper, the individual who purchases the item; ?store, the location where the item is purchased; ?bought, the item that is purchased; and ?elsewhere, the individual's next location after the store.

3.2 Activating the script

MicroSAM receives an input story as a list of events. Story events take the form of CD structures in the program implementation. However, for demonstrative purposes, we express stories here in natural language. Consider the following story:

Shopping story
1. Jane went to the store.
2. She got a kite.
3. She went home.

When processing an input event, MicroSAM determines if the event contains a reference to a script. A script reference is signaled by the appearance of the designated keyword for a script. Upon detection of a keyword, the respective script is activated. In event 1 of the example story above, MicroSAM processes the symbol STORE and consequently activates $SHOPPING. $SHOPPING becomes the active script. Each time it processes an input event, MicroSAM checks the event for a script reference. If a script reference is detected, the corresponding script is activated. If there is already an active script when a new script is referenced, the previous script is no longer active in favor of the new script.

3.3 Applying the script

3.3.1 Pattern matching

To determine the parts of the script that have occurred in the story, MicroSAM initiates a *pattern matching* process. When it receives an input event, it tests for equivalence between the CD structure of the event and the CD structure of the first available pattern in the active script. The test for equivalence is performed as a graph isomorphism over the two CD structures. If the two CD structures are equivalent, they are said to match. If there is no match, MicroSAM compares the event to the next pattern, and so on, until a match is found. A match suggests that the respective script pattern has occurred in the story.

Consider event 1 of the input story and pattern 1 of $SHOPPING:

Story, event 1: "Jane went to the store."
(ptrans (actor (Jane)) (object (Jane)) (to (store)))

Script, pattern 1: "Someone went to the store."
(ptrans (actor ?shopper) (object ?shopper) (to ?store))

MicroSAM determines that the CD structures of the event and pattern are equivalent. Consequently, we say that event 1 matches pattern 1. The event is added to MicroSAM's internal list of events representing its full working memory understanding of the story. In MicroSAM's code and associated literature, this list is called the *database*.

3.3.2 Script binding

As patterns are matched, MicroSAM tracks which variables in the script are filled by, or bound to, which values in the story. This is a process called *script binding*. (This term is also used to refer to the resulting mapping between variable and value.) Following the match above between event 1 and pattern 1, "Jane" is bound to the ?shopper variable, while "store" is bound to the ?store variable.

MicroSAM stores the script bindings in the database. Following the application process, the database contains a script combination, and the bindings list contains both bindings of script variables to constants.

3.3.3 Instantiation

In the example discussed, we determined that MicroSAM matches event 1 to pattern 1. Next, it matches event 2 ("She got a kite") to pattern 3 ("The store transfers possession of the object to her"). Notice that pattern 2 of the script ("She picks up an object") was not matched to a story event.

If MicroSAM matches a script pattern but does not match a prior pattern or patterns, the program integrates the skipped pattern(s) into the story understanding. In a process called *instantiation*, MicroSAM refers to the script bindings in the database to replace the variables in the skipped pattern with the respective values to which they are bound in the story. Then, in the process we refer to as *backfilling*, the pattern, now instantiated as an event, is added to the database in the order that it occurs with respect to the matched pattern in the script.

3.4 Output

The processes of matching, script binding, instantiation, and backfilling repeat until MicroSAM processes all events in the story. When the program finishes, it outputs the database representing its understanding of the story. This understanding includes both explicit events—events in the story that matched to patterns in the script—as well as implicit events—script patterns that were not in the story but instantiated with script bindings.

MicroSAM output

Jane went to a store.
Jane picked up a kite.
The store transferred possession of the kite to Jane.
Jane transferred possession of some money to the store.
Jane left the store.

4. Multiple Script Activator

We have set out to implement an enhancement of MicroSAM, called the Multiple Script Activator (MUSCRAT), that enables multiple scripts to be active at once.

Recall that a script in MicroSAM is activated when the program detects a keyword in an input event, signaling a reference to a script. The script continues to be active until a new script reference is found in the story, at which point the program activates the new script and discontinues activation of the previous one. Hence, only one script can be active at a time. Rather than discard a previously active script in favor of a newly activated one, MUSCRAT stores active scripts in a list. When a script is activated, it is added to the list. If a new script reference is found when another script is already active, the new script is simply appended to the list so that both scripts are active.

Each active script is stored in the list as its own data structure. The data structure contains the script's name, its script patterns, and its script bindings. When attempting to match an input story

$RESTAURANT	$BIRTHDAY	Restaurant-birthday Story
1. Diner goes to restaurant. 2. Diner says they want to eat meal. 3. Diner eats meal. 4. Diner gives tip to waiter. 5. Diner pays restaurant. 6. Diner leaves restaurant.	1. Guest wishes celebrant a happy birthday. 2. Guest gives gift to celebrant.	1. Mary goes to a restaurant. 2. Mary eats steak. 3. The waiter wishes Mary a happy birthday. 4. The waiter gives a cake to Mary. 5. Mary leaves the restaurant.

Figure 2. The $RESTAURANT and $BIRTHDAY scripts (in natural language) and the RESTAURANT-BIRTHDAY story. The scripts are shown in simplified English rather than conceptual dependency for ease of understanding.

event to a script pattern, MUSCRAT searches each active script sequentially until the first pattern that matches the event is found. For instance, if Scripts A and B are active, MUSCRAT will first search Script A's patterns for a match. If a match is found in Script A, the search terminates. If there is no match in Script A, MUSCRAT proceeds to search Script B.

As in the original MicroSAM, when there is a match, the matched pattern is added to the program's database. If there are patterns that were "skipped over" in the script, i.e. patterns in the script that precede the matched pattern but were not themselves matched, they are instantiated with the appropriate script bindings and added to the database as well. In MUSCRAT, all patterns that are added to the database are consequently removed from their respective script's data structure, so that at any point MUSCRAT attempts to match, it compares events only with script patterns that have not yet occurred in the story. Only the script in which a match is found is updated; all other active scripts are unchanged during an iteration in which they did not turn up a match.

4.1 Script Corpus

In order to implement and test a program that can apply multiple scripts, we conceptualized scenarios that draw upon more than one script. We consulted Schank and Abelson's $RESTAURANT script, then pursued the idea that often at restaurants, people celebrate birthdays. We decided to combine the restaurant script with what could be called the $BIRTHDAY script, or the typical sequence of events that occurs at a birthday celebration.

The $RESTAURANT script has several versions. One version is the original, provided from Schank and Abelson's *Scripts, Plans, Goals and Understanding* (1977). However, this version is quite long and complex, as it has multiple turning points at which events can vary depending on the instance.

Another version is the abridged $RESTAURANT script that was provided with the MicroSAM program. It contains five patterns, which describe going to the restaurant, ordering a meal, eating the meal, paying for the meal, and leaving the restaurant. In our research we use a modified version of this script that includes all of the aforementioned events, as well as an additional event from the original $RESTAURANT script in which the individual gives a tip to the waiter. The $RESTAURANT script is activated by the keyword “restaurant”. It is shown in Figure 2 in simplified English rather than Conceptual Dependency for ease of understanding.

To our knowledge there is no existing birthday script in the literature, thus we were tasked to create our own original take. First, we considered the events that typically occur at a birthday party and realized that birthday celebrations are liable to a great deal of variation. We minimized the chance for variation and included only two basic events: a guest wishes the celebrant a happy birthday and gives them a present. We converted these events into CD form and created the $BIRTHDAY script. The $BIRTHDAY script is activated by the keyword HAPPY-BIRTHDAY and is shown in Figure 2.

4.2 An Example

Having identified two distinct scripts that could combine within one story, we created an original story in order to test MUSCRAT's capabilities. Consider the RESTAURANT-BIRTHDAY story in Figure 2, which references the $RESTAURANT and $BIRTHDAY scripts.

We run the MUSCRAT program on the above story, as illustrated in Figure 3. We describe the process as follows. (Note that we use conceptual dependency structures, and that script variables are represented by symbols, mapped in the key on the bottom left.)

MUSCRAT processes event 1. A script reference is found: it detects the keyword RESTAURANT within the input CD structure and activates $RESTAURANT. Event 1 matches pattern 1 in $RESTAURANT. Event 1 is added to the database.

MUSCRAT processes event 2. No new script reference is detected. The active script, $RESTAURANT, is checked for a match. Event 2 matches pattern 3 in $RESTAURANT. This skips over pattern 2 in $RESTAURANT. Pattern 2 is instantiated with the appropriate script bindings, substituting Mary for ?DINER (symbolized by D) and steak for ?MEAL (symbolized by M), and added to the database. Event 2 is added to the database.

MUSCRAT processes event 3. A script reference is found: it detects the keyword HAPPY-BIRTHDAY and activates $BIRTHDAY. Event 3 matches pattern 1 in $BIRTHDAY. Event 3 is added to the database.

MUSCRAT processes event 4. No new script reference is found. The active scripts, $RESTAURANT and $BIRTHDAY, are checked for a match. Event 4 matches pattern 2 in $BIRTHDAY. Event 4 is added to the database.

MUSCRAT processes event 5. No new script reference is found. The active scripts, $RESTAURANT and $BIRTHDAY, are checked for a match. Event 5 matches pattern 6 in $RESTAURANT. This skips over patterns 4 and 5 in $RESTAURANT. Patterns 4 and 5 are instantiated with the appropriate script bindings, substituting Mary for ?DINER (D) and restaurant for ?RESTAURANT (R), and added to the database. Then, event 5 is added to the database.

At this point all input story events have been processed. MUSCRAT returns the database, representing the final understanding of the story.

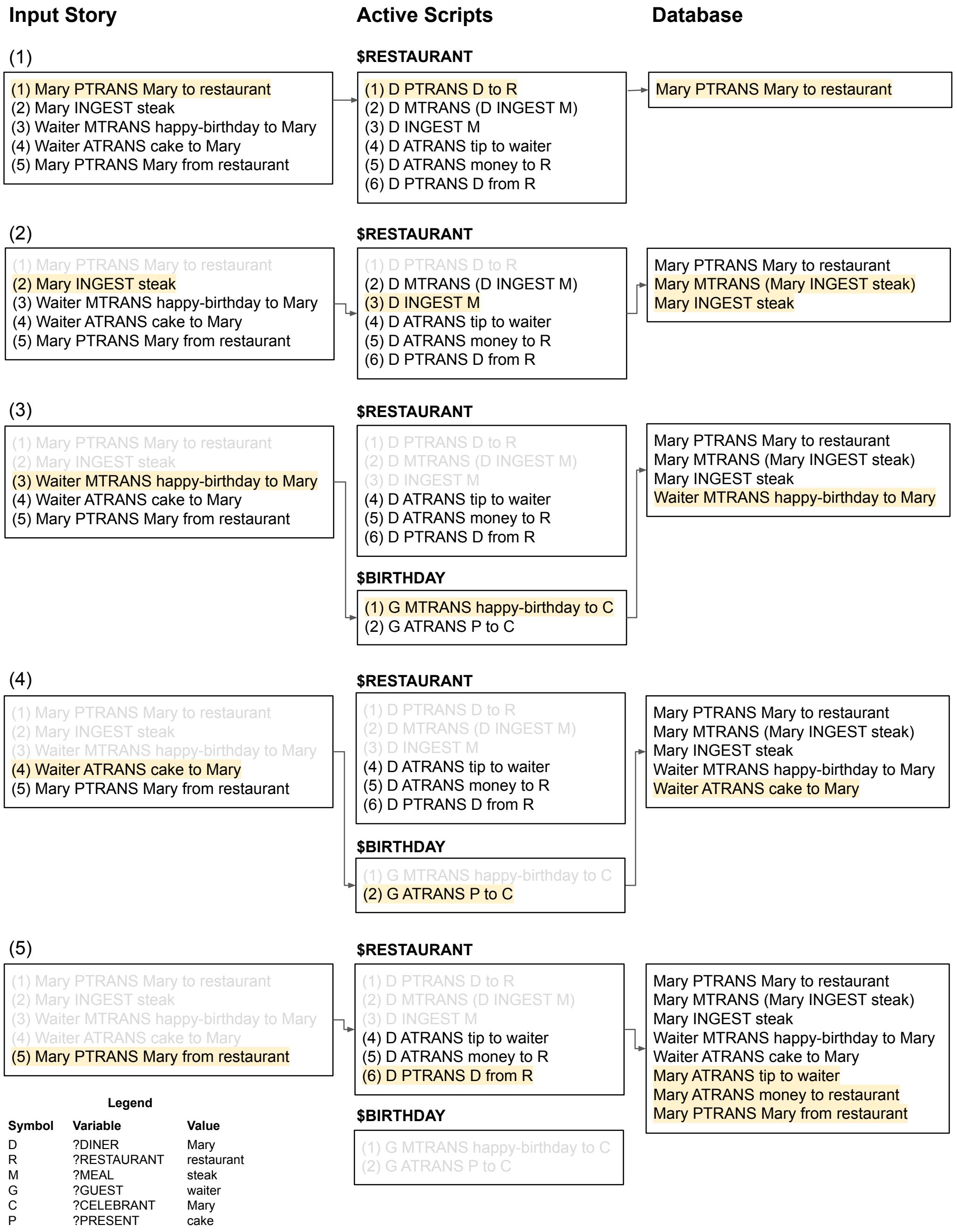

Input Story
Active Scripts
Database
(1)
$RESTAURANT
(1) Mary PTRANS Mary to restaurant
(2) Mary INGEST steak
(3) Waiter MTRANS happy-birthday to Mary
(4) Waiter ATRANS cake to Mary
(5) Mary PTRANS Mary from restaurant
(1) D PTRANS D to R
(2) D MTRANS (D INGEST M)
(3) D INGEST M
(4) D ATRANS tip to waiter
(5) D ATRANS money to R
(6) D PTRANS D from R
Mary PTRANS Mary to restaurant
(2)
$RESTAURANT
(1) Mary PTRANS Mary to restaurant
(2) Mary INGEST steak
(3) Waiter MTRANS happy-birthday to Mary
(4) Waiter ATRANS cake to Mary
(5) Mary PTRANS Mary from restaurant
(1) D PTRANS D to R
(2) D MTRANS (D INGEST M)
(3) D INGEST M
(4) D ATRANS tip to waiter
(5) D ATRANS money to R
(6) D PTRANS D from R
Mary PTRANS Mary to restaurant
Mary MTRANS (Mary INGEST steak)
Mary INGEST steak
(3)
$RESTAURANT
(1) Mary PTRANS Mary to restaurant
(2) Mary INGEST steak
(3) Waiter MTRANS happy-birthday to Mary
(4) Waiter ATRANS cake to Mary
(5) Mary PTRANS Mary from restaurant
(1) D PTRANS D to R
(2) D MTRANS (D INGEST M)
(3) D INGEST M
(4) D ATRANS tip to waiter
(5) D ATRANS money to R
(6) D PTRANS D from R
$BIRTHDAY
(1) G MTRANS happy-birthday to C
(2) G ATRANS P to C
Mary PTRANS Mary to restaurant
Mary MTRANS (Mary INGEST steak)
Mary INGEST steak
Waiter MTRANS happy-birthday to Mary
(4)
$RESTAURANT
(1) Mary PTRANS Mary to restaurant
(2) Mary INGEST steak
(3) Waiter MTRANS happy-birthday to Mary
(4) Waiter ATRANS cake to Mary
(5) Mary PTRANS Mary from restaurant
(1) D PTRANS D to R
(2) D MTRANS (D INGEST M)
(3) D INGEST M
(4) D ATRANS tip to waiter
(5) D ATRANS money to R
(6) D PTRANS D from R
$BIRTHDAY
(1) G MTRANS happy-birthday to C
(2) G ATRANS P to C
Mary PTRANS Mary to restaurant
Mary MTRANS (Mary INGEST steak)
Mary INGEST steak
Waiter MTRANS happy-birthday to Mary
Waiter ATRANS cake to Mary
(5)
$RESTAURANT
(1) Mary PTRANS Mary to restaurant
(2) Mary INGEST steak
(3) Waiter MTRANS happy-birthday to Mary
(4) Waiter ATRANS cake to Mary
(5) Mary PTRANS Mary from restaurant
(1) D PTRANS D to R
(2) D MTRANS (D INGEST M)
(3) D INGEST M
(4) D ATRANS tip to waiter
(5) D ATRANS money to R
(6) D PTRANS D from R
$BIRTHDAY
(1) G MTRANS happy-birthday to C
(2) G ATRANS P to C
Mary PTRANS Mary to restaurant
Mary MTRANS (Mary INGEST steak)
Mary INGEST steak
Waiter MTRANS happy-birthday to Mary
Waiter ATRANS cake to Mary
Mary ATRANS tip to waiter
Mary ATRANS money to restaurant
Mary PTRANS Mary from restaurant
Legend
Symbol Variable Value
D ?DINER Mary
R ?RESTAURANT restaurant
M ?MEAL steak
G ?GUEST waiter
C ?CELEBRANT Mary
P ?PRESENT cake

Figure 3. The Multiple Script Activator is an enhancement of MicroSAM that can reference multiple scripts in order to process an input story. Each event in the input story is checked for a match to a pattern in one of the active scripts. Events that are matched are added to the database, as well as any script patterns that are skipped over. The variables in skipped script patterns are instantiated with their respective values.

5. Story Generation through Script Combination

There has been limited work on story generation with scripts (Ogata et al., 2016; Li et al., 2013). However, none of the prior work focuses on creating stories by combining scripts using rich combinations of conceptual primitives, actor, object, and instrument roles. Our second novel script combinator system, which we call the Script Combination Applier Mechanism (SCAM), enhances the Script Applier Mechanism so that a script can be "applied" to another script, combining the two scripts together.

Instead of receiving a story as input, and activating and applying a script to the story events, SCAM receives two scripts as input. SCAM uses the same mechanisms as MicroSAM, with the exception that it treats one of the scripts as if it were a story, and then it activates and applies the other script to it.

To match the scripts to each other, similar to MicroSAM, SCAM determines the parts of one script that occur in the other, initiating a pattern matching process starting with the first event in each of the two scripts. SCAM compares the CD structure of the event to that of the first available pattern in the active script. There is a match if the two CD structures are equivalent. If there is no match, SCAM compares the pattern in the first script to the next pattern in the second script, and so on, until a match is found.

The main modification is in the MATCHer component of script application, which, as with the multiple script activator story understanding system, performs a matching as a graph isomorphism over CD structures. The MATCH function is modified so that, instead of accepting a "constant" CD structure and a script "pattern" CD structure, it now accepts two "pattern" CD structures containing script variables. Again, subgraphs that are present in one structure and not in the other are ignored in the matching.

As with the original MATCHer, when performing the isomorphic match, "constant" CD structure elements are matched against constants, and variables are matched against constants using their bindings, resulting in a binding for the variable if it is not already bound. Because both inputs to the matcher are patterns, both contain variables, and the matcher now supports matching variables against variables. In these cases, the script variables end up bound to each other instead of being bound to non-variable "constant" components of the structure.

Just as with MUSCRAT, following the application process, the "database" contains a script combination, and the bindings list contains both bindings of script variables to constants, and bindings of script variables to other script variables.

5.1 An Example

In the example shown in Figure 4, the simplified restaurant script from the previous section is combined with the simplified birthday party script. In the MUSCRAT story understanding example (Section 4.2) the process of having multiple scripts active was invoked and guided by the story conceptualization. In contrast, because in this case the script combination is not guided by a story, there are combinatorially many more possible combinations of the script CD structures. This example illustrates one possible combination of these two scripts.

In this combination, the ATRANS of the ?gift in the birthday script is matched against the ATRANS of the tip in the restaurant script rather than the ATRANS of the cake. This has the effect of binding the restaurant waiter as the birthday ?celebrant, the restaurant ?customer as the ?guest, and the tip as the ?gift. This is followed by the "backfill" operation, which adds the MTRANS of the "happy birthday" greeting in working memory (the database) before the ATRANS of the ?gift. The MTRANS of the greeting also has the same bindings: the restaurant ?customer, who is also the birthday ?guest, is the ACTOR of the MTRANS, and the restaurant ?waiter, who is also the birthday ?celebrant, is the recipient of the greeting MTRANS.

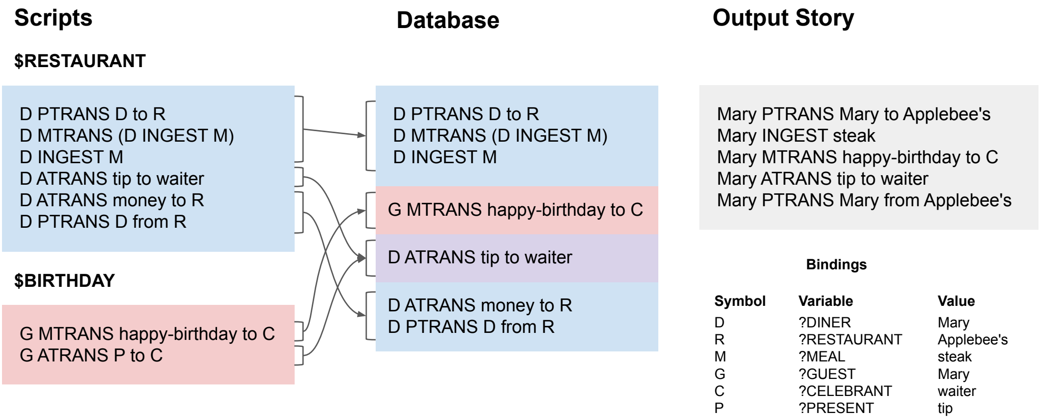
Put differently, this script combination has reversed the roles with respect to the example in Section 4.2, making it so that it is the waiter's birthday (while on the job) and the customer is a guest at a birthday party at the restaurant. The restaurant customer, in addition to doing the usual things that a restaurant customer does, also wishes the waiter a happy birthday, and pays the waiter a tip as a birthday present.

The script combination at this point is not sufficient to use for story generation, because components of the combined script structure are unbound variables, or variables that are bound to other variables. Figure 4 shows how a story could be generated from the script combination by assigning concrete structures to the variables. An English surface realization of the story could be:

> Mary went to Applebee's. After having a steak, she wished her waiter a happy birthday. Mary gave him a tip as a present and left.

6. Discussion

The purpose of MicroSAM is to fill in script events that are not explicitly mentioned in an input story. With the added benefit of MUSCRAT, more than one script can be consulted when processing



Symbol	Variable	Value
D	?DINER	Mary
R	?RESTAURANT	Applebee's
M	?MEAL	steak
G	?GUEST	Mary
C	?CELEBRANT	waiter
P	?PRESENT	tip

Figure 4. An example of the Script Combination Applier Mechanism combining a $RESTAURANT and a $BIRTHDAY script together. This example celebrates the waiter's birthday, and has the ?DINER giving a tip as the birthday present. Following the combination, script variables are bound to other script variables: ?DINER is bound to ?GUEST.

a story in order to determine what event or events are "missing." In MUSCRAT, missing events are appended to the end of the database in the order that they appear in the script, immediately followed by the matched event.

One challenge with this choice of implementation is that we face the issue where events may be added to the database in an order that violates what humans may perceive as common sense. It is possible that an event that is instantiated from one script and added to the database matches an event that has already been added from a different script. Currently, MUSCRAT has no way of controlling duplicate structures within the database. Additionally, the database is a sequential order of story events. It is possible that an event added to the database should logically appear at an earlier point in the database than where it was added. MUSCRAT is unable to reason about or re-order events within the database. Another issue posed by MUSCRAT is that, as was implemented in MicroSAM, scripts are activated by a single keyword that we determine to be a likely indication of a script. For example, $RESTAURANT is activated by the keyword RESTAURANT and $BIRTHDAY by the keyword HAPPY-BIRTHDAY. However, not every instance in which a restaurant or the words "happy birthday" appear in a story is an instance in which the restaurant and birthday scripts, respectively, are called upon.

Both the MUSCRAT and SCAM examples we have presented illustrate activating or combining scripts which generally represent very different situations. Regarding story understanding, one of the main critiques of scripts traditionally is that a script-based understander will have difficulty processing unexpected events that do not fit the activated script. MUSCRAT hints at the possibility that, by activating multiple scripts, a story understander may be able to process events that are unexpected in a script by simply activating another script which is a better match for that event, and allowing both scripts to remain active. SCAM and its example show how combining dissimilar scripts can create story structures that are novel and interesting because they have events that come from one script which are unexpected in the other, as well as novel and interesting role binding combinations (e.g., tipping the waiter as a birthday present).

Earlier work on dynamic memory systems (e.g., Lebowitz, 1983; Kolodner, 1981) demonstrated how scripts evolve in the presence of new concrete episodic memories which have events which match a script, and novel unexpected events that don't match, but lead to script generalizations. Although our SCAM example illustrates a situation where two scripts that are somewhat unlike one another get matched together, a SCAM-like process could also be used to combine similar scripts together (e.g., $DENTIST and $DOCTOR) to form a generalization (e.g., $HEALTH-CARE-VISIT). SCAM illustrates how generalizations may be able to occur entirely within the space of scripts, without a concrete episodic memory structure being present when scripts are applied to one another.

One obvious but important way in which SCAM differs from MUSCRAT is that unassigned script variables play a wider role in the structures. In this regard SCAM would appear to be on a more direct path toward abstract, high-level thinking and reasoning, because it creates structures where variables can be bound to other variables in a script, and structures where variables are more likely to be unbound.

7. Conclusion and Future work

In this paper we described the MUSCRAT and SCAM systems, which re-examine original work on scripts and explore how, by combining scripts, it is possible to increase the natural language understanding and generation capabilities of script-based natural language systems. In understanding, we showed how this allows for flexible comprehension of new situations through an example of a birthday party in a restaurant. In generation, we showed how the combination mechanism enables the generation of diverse stories. In both of these cases, we discussed design decisions and issues that arise from them. Our work is unique in that it takes steps toward defining script application and combination as general cognitive processes.

In future work on both MUSCRAT and SCAM, we will conduct studies on activating and combining more than two scripts, and using multiple script processing to deal with common assumptions (e.g., combining with a $WALKING script to represent the assumption that story characters PTRANSing themselves is instrumented by moving their legs). Connected to this work is the project of building a large repository of scripts using primitive decomposition systems as the event representations. Systems which have been used for crowdsourcing scripts using other event representations (e.g., Ciosici et al., 2021) may be adapted to this purpose. Also, in our examples we have ignored the complex issues of how to generate stories well from script combinations. Given that scripts are meant to represent “mundane” information about situations, a particularly interesting issue is determining what events from a script combination are important to include in telling a story, and what events to leave out. Future work in which human participants read and react to stories will determine whether events that have been “merged” between two scripts carry greater importance in composing an understandable story. We intend for better demonstrations of story generation which link SCAM to a natural language generation system which generates surface realizations from primitive decomposed representations like CD.

We recognize that there are significant challenges to using a primitive decomposition system like Conceptual Dependency as the representation system for scripts. The natural concern is that representing concepts using a small number of very abstract and general conceptual primitives instead of language “oversimplifies” or represents a loss of detail. Both this problem and the “inference explosion” problem observed in the MARGIE system (see Schank & Riesbeck, 1982) can be overcome by realizing that primitives can be combined in sophisticated ways in the representation of a concept to represent greater detail; performing matches and inferences on larger structures built from primitives can reduce explosions and the "everything matching everything" problem. In part, scripts were invented as epitomies of this view when they are represented as chains of primitive decomposed events and states (Schank & Abelson, 1977).

One of the critiques of scripts was their inflexibility in dealing with reasoning about goals and plans, thus leading to the creation of separate goal and plan structures in Schank and Abelson’s original treatise (Schank & Abelson, 1977). We suspect that the reason for this choice was due to a lack of prior work in how Conceptual Dependency might have represented concepts related to goals, plans, expectations, and decision making. Taking a broad view of scripts as packaged episodic memories, we see no reason why episodic memories of goal reasoning or planning processes could not be represented as scripts—perhaps using CD primitives such as MBUILD, MLOC, or MTRANS—and combined with other scripts when needed to understand unexpected events. Researching high-level memory organization structures like scripts appears to go hand in hand with researching their underlying decomposed representations.

The work that followed scripts (Schank, 1983) explored mechanisms in which scripts are related to other scripts at a "knowledge structure" level. We are curious as to whether script combinations, or some variation of them can or play an important role in the formation of memory organization packets at the knowledge structure level. This will be important work toward the goal of unifying scripts with higher-level knowledge structures to allow for script activation and combination to be applicable more broadly. We hope that our studies of novel forms of script application and manipulation will revive interest in scripts and how these structures, as carriers of Schank and Abelson's original idea of generalized, standardized memory episodes, could be posed as an alternative or an addition to logical formulations of knowledge representation and reasoning.

References

Boujarwah, F., Abowd, G., & Arriaga, R. (2012). Socially computed scripts to support social problem solving skills. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1987–1996). ACM.

Bower, G. H., Black, J. B., & Turner, T. J. (1979). Scripts in memory for text. *Cognitive psychology*, *11*, 177–220.

Chambers, N. (2017). Behind the scenes of an evolving event cloze test. *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics* (pp. 41–45).

Chambers, N., & Jurafsky, D. (2008). Unsupervised learning of narrative event chains. *ACL-08* (pp. 789–797). Columbus, Ohio, USA.

Chambers, N., & Jurafsky, D. (2009). Unsupervised learning of narrative schemas and their participants. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2* (pp. 602–610). Suntec, Singapore: Association for Computational Linguistics.

Ciosici, M., Cummings, J., DeHaven, M., Hedges, A., Kankanampati, Y., Lee, D.-H., Weischedel, R., & Freedman, M. (2021). Machine-assisted script curation. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations* (pp. 8–17). Online: Association for Computational Linguistics.

Cullingford, R. E. (1977). *Script application: computer understanding of newspaper stories.*. Yale University.

Dyer, M. G. (1982). *In-depth understanding: A computer model of integrated processing for narrative comprehension*. Yale University.

Granger, R. H. (1980). *Adaptive understanding: Correcting erroneous inferences*. Technical Report Research Report #171, Yale University, Department of Computer Science, New Haven, CT.

Kolodner, J. L. (1981). Organization and retrieval in a conceptual memory for events or con54, where are you? *Proceedings of the Seventh International Joint Conference on Artificial Intelli-*

gence (pp. 227–233). Vancouver, BC.

Lebowitz, M. (1983). Memory-based parsing. *Artificial Intelligence*, *21*, 363–404.

Lehnert, W. G. (1977). *The process of question answering.*. Yale University.

Li, B., Appling, D. S., Lee-Urban, S., & Riedl, M. O. (2012). Crowdsourcing narrative intelligence. *Advances in Cognitive Systems*, *2*, 25–42.

Li, B., Lee-Urban, S., Johnston, G., & Riedl, M. (2013). Story generation with crowdsourced plot graphs. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 598–604).

Macbeth, J. C., Gromann, D., & Hedblom, M. M. (2017). Image schemas and conceptual dependency primitives: A comparison.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.

Mueller, E. T. (2004). Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research*, *5*, 307–340.

Ogata, T., Arai, T., & Ono, J. (2016). Using synthetically collected scripts for story generation. *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations* (pp. 253–257). Osaka, Japan: The COLING 2016 Organizing Committee. From `https://aclanthology.org/C16-2053`.

Rudinger, R., Rastogi, P., Ferraro, F., & Van Durme, B. (2015). Script induction as language modeling. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1681–1686). Lisbon, Portugal: Association for Computational Linguistics. From `https://aclanthology.org/D15-1195`.

Schank, R. C. (1972). Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, *3*, 552–631.

Schank, R. C. (1975). *Conceptual information processing*. New York, NY: Elsevier.

Schank, R. C. (1983). *Dynamic memory: A theory of reminding and learning in computers and people*. New York: Cambridge University Press.

Schank, R. C., & Abelson, R. P. (1975). Scripts, plans, and knowledge. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (pp. 151–157). Tbilisi, Georgia.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Mahwah, NJ: Lawrence Erlbaum Associates.

Schank, R. C., & Burstein, M. (1982). *Modeling memory for language understanding*. Technical Report Research Report #220, Yale University, Department of Computer Science, New Haven, CT.

Schank, R. C., & Riesbeck, C. K. (1982). *Inside computer understanding: Five programs plus miniatures*. Hillsdale, NJ: L. Erlbaum Associates Inc.

Slade, S. (1987). The yale artificial intelligence project: A brief history. *AI Magazine*, *8*, 67–67.

Weber, N., Rudinger, R., & Van Durme, B. (2020). Causal inference of script knowledge.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 7583–7596). Online: Association for Computational Linguistics. From `https://aclanthology.org/2020.emnlp-main.612`.

Weber, N., Shekhar, L., Balasubramanian, N., & Chambers, N. (2018). Hierarchical quantized representations for script generation. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 3783–3792). Brussels, Belgium: Association for Computational Linguistics. From `https://aclanthology.org/D18-1413`.

Winograd, T. (1978). On primitives, prototypes, and other semantic anomalies. *Proceedings of the 1978 Workshop on Theoretical Issues in Natural Language Processing* (pp. 25–32). Stroudsburg, PA, USA: Association for Computational Linguistics.