# Script Knowledge for Natural Language Understanding

Dissertation

zur Erlangung des akademischen Grades

eines Doktors der Philosophie

an den Philosophischen Fakultäten

der Universität des Saarlandes

vorgelegt von

Simon Ostermann

aus Neunkirchen

Saarbrücken, 2020

Dekan der Philosophischen Fakultät: Prof. Dr. Heinrich Schlange-Schöningen

Erstgutachter: Prof. Dr. Manfred Pinkal

Zweitgutachter: Prof Dr. Alexander Koller

Drittgutachter: Dr. Michael Roth

Tag der letzten Prüfungsleistung: 19. Dezember 2019

## Abstract

While people process text, they make frequent use of information that is assumed to be common ground and left implicit in the text. One important type of such commonsense knowledge is script knowledge, which is the knowledge about the events and participants in everyday activities such as *visiting a restaurant*. Due to its implicitness, it is hard for machines to exploit such script knowledge for natural language processing (NLP). This dissertation addresses the role of script knowledge in a central field of NLP, natural language understanding (NLU).

In the first part of this thesis, we address script parsing. The idea of script parsing is to align event and participant mentions in a text with an underlying script representation. This makes it possible for a system to leverage script knowledge for downstream tasks. We develop the first script parsing model for events that can be trained on a large scale on crowdsourced script data. The model is implemented as a linear-chain conditional random field and trained on sequences of short event descriptions, implicitly exploiting the inherent event ordering information. We show that this ordering information plays a crucial role for script parsing. Our model provides an important first step towards facilitating the use of script knowledge for NLU.

In the second part of the thesis, we move our focus to an actual application in the area of NLU, i.e. machine comprehension. For the first time, we provide data sets for the systematic evaluation of the contribution of script knowledge for machine comprehension. We create MCScript, a corpus of narrations about everyday activities and questions on the texts. By collecting questions based on a scenario rather than a text, we aimed at creating challenging questions which require script knowledge for finding the correct answer. Based on the findings of a shared task carried out with the data set, which indicated that script knowledge is not relevant for good performance on our corpus, we revised the data collection process and created a second version of the data set.

# Kurzzusammenfassung

Wenn Menschen Text verarbeiten, machen sie häufig Gebrauch von Information, die als allgemeingültig angenommen und daher im Text nicht realisiert wird. Eine wichtige Art solchen Allgemeinwissens ist das Skriptwissen, das heißt das Wissen über die Ereignisse und Partizipanten von Alltagsszenarien, wie z.B. ein *Restaurantbesuch*. Aufgrund seiner Impliziteit ist es für Maschinen anspruchsvoll, solches Wissen bei der Verarbeitung von Text zu verwenden. Diese Dissertation behandelt die Rolle von Skriptwissen für einen wichtigen Teilbereich der Computerlinguistik, nämlich das automatische Verstehen natürlicher Sprache (Natural Language Understanding, NLU).

Im ersten Teil der Dissertation behandeln wir das Skript-Parsing. Die Idee des Skript-Parsing ist es, Ereignis- und Partizipant-Erwähnungen in einem Text mit einer zugrundeliegenden Skriptrepräsentation zu alignieren, damit diese explizit für Anwendungen verfügbar gemacht werden kann. Wir stellen den ersten Skript-Parser für Ereignisse bereit, der im großen Stil auf durch Crowdsourcing gewonnenen Skriptdaten trainiert werden kann. Implementiert wird das Modell als Linear-Chain Conditional Random Field, welches auf Sequenzen von kurzen Ereignisbeschreibungen trainiert wird und implizit die temporale Abfolge dieser Ereignisse lernt. Wir zeigen, dass solche temporale Ordnungsinformation essentiell für das Skript-Parsen ist. Unser Modell stellt einen wichtigen ersten Schritt zur Bereitstellung von Skriptwissen für NLU dar.

Im zweiten Teil der Dissertation legen wir den Fokus auf eine Anwendung im Bereich NLU: Machine Comprehension, eine Leseverstehensaufgabe, bei der Systeme einen Text verarbeiten und Fragen zu dem Text beantworten müssen. Zum ersten Mal stellen wir Daten zur systematischen Evaluierung des Einflusses von Skriptwissen auf Machine Comprehension bereit. Wir erstellen MCScript, ein Corpus aus narrativen Texten über Alltagsszenarien und Fragen zu den Texten. Indem wir die Fragen auf Basis eines Szenarios statt eines Textes sammeln, versuchen wir schwierige Fragen zu erzeugen, die Skriptwissen zum Finden einer Antwort benötigen. Die Ergebnisse eines Shared Tasks, den wir auf den Daten durchgeführt haben, legen jedoch nahe, dass Skriptwissen nicht vonnöten ist, um auf den Daten eine gute Performanz zu erreichen. Infolgedessen revidieren wir den Datenerzeugungsprozess und erstellen eine zweite Version des Datensatzes.

# Ausführliche Zusammenfassung

Wenn Menschen Text verarbeiten, machen sie häufig Gebrauch von Information, die als allgemein gültig angenommen und daher im Text nicht realisiert wird. Eine wichtige Art solchen Allgemeinwissens ist das **Skriptwissen**. Skripte umfassen das Wissen über die Ereignisse und Partizipanten von Alltagsszenarien, wie z.B. *Restaurantbesuch* oder *einen Kuchen backen*.

(1) Person A: „Gestern bin ich zu diesem neuen argentinischen Restaurant zum Abendessen gegangen. Ich hatte ein leckeres Steak."
Person B: „Was hast du bezahlt?"

Skriptwissen macht Alltagskommunikation zwischen Menschen erst möglich und gleichzeitig hocheffizient. Die Frage in dem Beispiel macht Sinn, obwohl Person A nie erwähnt hat, dass sie bezahlt hat. B kann sowohl inferieren, dass A Essen bestellt und bekommen hat, als auch dass sie vorm Verlassen des Restaurants gegessen und bezahlt hat, obwohl keines dieser Ereignisse erwähnt ist. Zusätzlich kann B folgern, welche Personen eine Rolle gespielt haben: Ein Ober hat vermutlich das Essen gebracht, welches von einem Koch zubereitet worden ist. A nimmt an, dass bestimmte Fakten allgemein für den Hörer bekannt sind (*common ground*, Stalnaker (2002)), was zum Beispiel das Skriptwissen über das Szenario *in ein Restaurant gehen* umfasst. Wenn sie die erste Äußerung hört, wird Person B eine mentale Repräsentation dieses Skriptes aktivieren, welche die erwähnte Information verfügbar macht.
Aufgrund der Implizitheit von Skriptwissen ist es für Maschinen nicht trivial, solches Wissen bei der Verarbeitung von Text zu verwenden. Diese Dissertation behandelt die Rolle von Skriptwissen für einen wichtigen Teilbereich der Computerlinguistik, das automatische Verstehen natürlicher Sprache (Natural Language Understanding, NLU).
Skriptwissen umfasst verschiedene Arten von Information:

- **Ereignisse (Events).** Ereignisse sind die Hauptkomponenten eines Skriptes. Sie beschreiben die einzelnen Schritte, die bei der Durchführung eines Skripts ausgeführt werden, wie zum Beispiel *bestellen* oder *bezahlen* im Kontext des *ein Restaurant besuchen* - Skripts.

- **Partizipanten.** Skriptwissen beinhaltet Informationen über die Personen und Dinge, die für ein Szenario eine Rolle spielen. Diese Rollen werden von konkreten Teilnehmern instanziiert, zum Beispiel von Personen wie dem *Ober* oder *Koch*, aber auch von Gegenständen wie *Essen*, *Besteck* oder *Geld*.
- **Temporale Ordnungsinformation.** Skriptereignisse sind temporal geordnet. *Bestellen* passiert zum Beispiel in der Regel vor dem *Essen*, welches wiederum vor dem Ereignis *zahlen* passiert.

Im Rahmen dieser Dissertation wird eine Skriptrepräsentation in der Form von Paraphrasenmengen nach Regneri et al. (2010) zugrunde gelegt. In dieser Repräsentation werden Skript-Ereignisse durch Mengen von kurzen Beschreibungen dargestellt, die Formulierungsalternativen für das Event bieten.

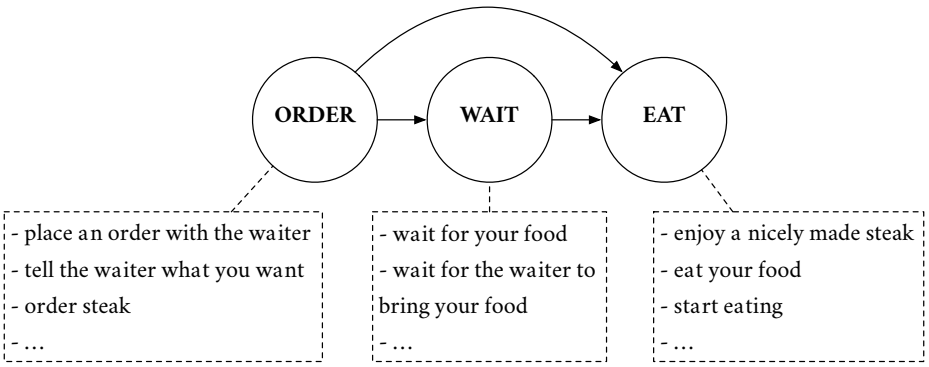



**Abbildung 1:** Ein Beispiel-Graph für das *ein Restaurant besuchen* - Skript.

Abbildung 1 zeigt einen Beispiel-Graphen für das *Restaurantbesuch* - Skript. Ereignisse werden durch die Knoten symbolisiert. Jedes Ereignis ist mit einer Paraphrasenmenge assoziiert: *Warten* beispielsweise wird durch die Ereignisbeschreibungen *wait for your food*, *wait for the waiter to bring your food* etc. dargestellt. Die Kanten zwischen den Knoten stellen die typische temporale Abfolge dar: *Bestellen* passiert (üblicherweise) vor *warten*.

Im Rahmen dieser Arbeit benutzen wir ein spezielles Skript-Corpus: *DeScript* (Wanzare et al., 2016) ist ein Corpus mit Skriptdaten zu einer großen Anzahl unterschiedlicher Alltagsszenarien. Das Corpus enthält sogenannte Eventsequenzbeschreibungen (*Event Sequence Description, ESD*), d.h. kurze Sequenzen von einzelnen Eventbeschreibungen im Telegrammstil, die einen „Durchlauf“eines Skripts beschreiben. ESDs in DeScript wurden durch Webexperimente erstellt, bei denen eine große Anzahl von Teilnehmern einfache Aufgaben für ein

geringes Entgelt erledigen. Diese Technik wird *Crowdsourcing* genannt und spielt auch für diese Arbeit eine zentrale Rolle. Teilnehmer an Crowdsourcing-Experimenten werden im Folgenden als *Arbeiter* (von engl. „worker“) bezeichnet.

Um eine Skriptrepräsentation aus Paraphrasenmengen auf Grundlage solcher ESDs zu erzeugen, können einzelne Eventbeschreibungen (EDs, d.h. Beschreibungen von einzelnen „Schritten“) unter Zuhilfenahme von temporaler Ordnungsinformation automatisch zu Paraphrasenmengen geclustert werden (Regneri et al., 2010; Wanzare et al., 2017). Für 10 der Szenarien in DeScript wurden Gold-Paraphrasenmengen für Ereignisse händisch erstellt. Jede Paraphrasenmenge ist zudem mit einer sprechenden Bezeichnung versehen, welche den Event-Typ indiziert.

**Skript-Parsing in natürlichen Texten.** Im ersten Teil dieser Dissertation betrachten wir die Aufgabe des *Skript-Parsing*, eine grundlegende Voraussetzung dafür, Skriptwissen für Anwendungen zugänglich zu machen. Die Idee des Skript-Parsens ist es, die Ereignis- und Partizipant-Labels eines Skripts auf den Text abzubilden, der verarbeitet werden soll. Durch diese Alignierung kann Skriptwissen zugänglich gemacht werden, das nicht explizit im Text realisiert wurde. Ein Skriptparser soll dabei nur auf der Skriptrepräsentation trainiert werden, um auf beliebigen Texten angewandt werden zu können.

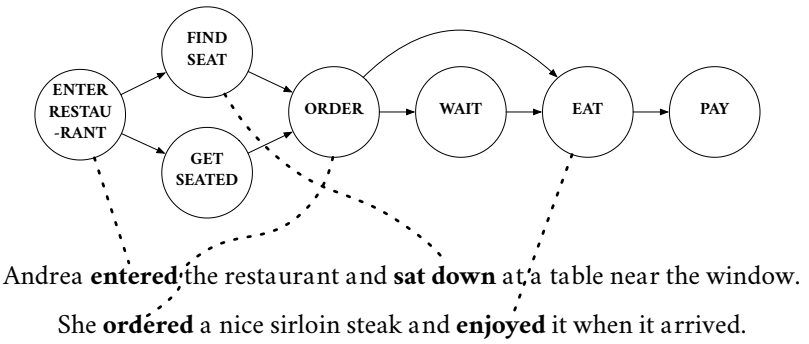



**Abbildung 2:** Ein Beispiel für Ereignis-basiertes Skript-Parsing.

Abbildung 2 zeigt ein Beispiel für Skript-Parsing auf Ereignissen. Die gestrichelten Linien stellen die Abbildung von Ereignissen auf ihre Instanzen in der Restaurant-Erzählung dar. Wenn ein System imstande ist, *enjoyed* mit dem *eat*-Ereignis zu alignieren, kann es aus Skriptwissen folgern und beispielsweise vorhersagen, dass das Ereignis *waiting* passiert ist, auch wenn es im Text nicht realisiert ist.

*Datensammlung.* In einem ersten Schritt befassen wir uns damit, Evaluationsdaten für unser später vorgestelltes Skript-Parsing-Modell bereitzustellen. Wir präsentieren InScript, ein Corpus aus Geschichten zu Alltagsereignissen, wie z.B. *Busfahren*, *einen Baum pflanzen* oder *einen Kuchen backen*. Jeder Text bezieht sich auf genau ein Szenario. Das Corpus wurde durch Crowdsourcing gewonnen und enthält 910 Texte zu den 10 Szenarien, zu denen es Gold-Paraphrasenmengen in DeScript gibt.

Zusammen mit den Texten stellen wir eine volle Ereignis- und Partizipantentypannotation bereit. Jedes Verb wurde händisch mit einem Eventtyplabel und jedes Substantiv mit einem Partizipantentyplabel versehen. Die verwendeten Labels basieren auf manuell erzeugten, szenario-spezifischen Templates. Die Labels sind identisch mit den Eventtyp-Labels in DeScript, der dieser Arbeit zugrunde liegenden Skriptrepräsentation. So kann jedem verwendeten Eventtyplabel in InScript eine Paraphrasenmenge in DeScript zugeordnet werden. Auf diese Weise stellt die Annotation einen Goldstandard für unser Skriptparsing-Modell dar, welches auf DeScript trainiert und auf InScript evaluiert werden kann.

*Datenanalyse.* Weil unsere Arbeit den ersten direkten Versuch des Skriptparsings darstellt, war es zu Beginn weitgehend unklar, wie schwierig die Aufgabe ist, und welche Art der linguistischen Information von Relevanz ist. Um dies abzuschätzen, präsentieren wir die Ergebnisse einer weiteren Annotationsstudie. In dieser Studie wurden Paare von Event-Instanz in InScript mit entsprechender Paraphrasenmenge in DeScript verglichen und mit einem Label versehen, welches die semantische Beziehung zwischen den beiden darstellt, wie z.B. *Synonymie*, *Entailment* oder *Inferenz*.

Eine Analyse der Resultate zeigt, dass Paraphrasenmengen die Schwierigkeit des Skriptparsens massiv reduzieren. Mit steigender Größe dieser Mengen steigt die Wahrscheinlichkeit, dass das gleiche Verb zum Beschreiben des Events im Text und in der Skriptrepräsentation benutzt wird, wodurch die Alignierung trivial ist.

*Skriptparsing-Modell.* Ein zentraler Bestandteil dieser Arbeit ist die Realisierung eines Modells zum Skriptparsing. Das präsentierte Modell ist der erste skalierbare Skriptparser, der auf beliebigen Paraphrasenmengen trainiert werden kann. Das Modell ist als Linear-Chain Conditional Random Field implementiert, welches auf ESDs trainiert wird. Das Modell benutzt implizit die temporale Ordnungsinformation von Ereignissen.

Wir zeigen, dass die Ordnungsinformation einen essentiellen Beitrag zur Performanz un-

| | |
|---|---|
| **T** | I wanted to plant a tree. I went to the home and garden store and picked a nice oak. Afterwards, I planted it in my garden. |
| **Q1** | What was used to dig the hole? |
| | a. a shovel b. his bare hands |
| **Q2** | When did he plant the tree? |
| | a. after watering it b. after taking it home |

**Abbildung 3:** Ein Beispiel für einen kurzen Text mit zwei Leseverstehensfragen.

seres Modells liefert, und dass das Modell Baseline-Systemen ohne Ordnungsinformation überlegen ist. Auf Grundlage der Entailment-Annotationen zeigen wir außerdem, dass unser Parser in der Tat von großen Paraphrasenmengen profitiert, aber dass ein großer Teil der Alignment-Fälle komplexe Inferenz erfordert, die von unserem Parser nur teilweise modelliert werden kann.

**Skriptwissen für Maschinelles Leseverstehen.** Im zweiten Teil der Dissertation legen wir den Fokus auf eine konkrete Anwendung im Bereich NLU: Maschinelles Leseverstehen (eng. Machine Comprehension). Maschinelles Leseverstehen ist eine Leseverstehensaufgabe, bei der Systeme einen Text verarbeiten und Fragen zu dem Text beantworten müssen. Bisher wurde Skriptwissen hauptsächlich in technischen Evaluierungen getestet, d.h. durch den Vergleich mit einem Skript-Goldstandard. Wir stellen zum ersten Mal einen Evaluationsrahmen für Skripte vor, der auf einer konkreten Aufgabe im Bereich NLU basiert.

Abbildung 3 illustriert die Idee. Gezeigt ist ein Textausschnitt mit zwei Leseverstehensfragen und je zwei Antwortkandidaten. Für Menschen ist es trivial, die Antwort auf beide Fragen zu finden: Eine Schaufel wird in der Regel dazu genutzt, Löcher zu graben; und ein Baum wird erst nach dem Wässern gepflanzt und nicht vor dem Kaufen. Diese Informationen sind nicht im Text gegeben, können aber aus dem Skriptwissen über das Szenario *einen Baum pflanzen* erschlossen werden. Für ein System muss solche Information also aus Hintergrundwissen erschlossen werden, da sie nicht aus dem Text entnommen werden kann.

*MCScript.* Als Teil dieser Arbeit haben wir *MCScript* entwickelt, den ersten Datensatz, der den Beitrag von Skriptwissen für Maschinelles Leseverstehen bewerten soll. MCScript ist ein Corpus aus 2.100 narrativen Texten zu 110 Szenarien. Ähnlich wie in InScript sind Tex-

te in MCScript kurze Geschichten zu je einem spezifischen Szenario. Zu den Texten gibt es insgesamt knapp 14.000 Fragen mit je zwei Antwortkandidaten. Ca. 27% der Fragen erfordern die Benutzung von Skriptwissen (im Folgenden als *Skript-basierte* Fragen bezeichnet).

Der komplette Datensatz wurde mit dem Crowdsourcing-Verfahren gesammelt. Texte haben wir nach demselben Verfahren wie bei der Erstellung von InScript erzeugt. Anspruchsvolle Fragen, die die Benutzung von Skriptwissen erfordern, wurden auf Basis einer neuen Methode der Fragesammlung erzeugt. Während der Fragesammlung haben wir den Arbeitern keinen konkreten Text gezeigt, sondern nur eine kurze Beschreibung des Szenarios. Die Arbeiter wurden dann dazu angeleitet, zu typischen Ereignissen und Partizipanten eines Szenarios Fragen zu stellen. Die so erzeugten Fragen wurden zufällig mit Texten des gleichen Szenarios gepaart. Anschließend wurde in einem dritten Crowdsourcing-Experiment eine Antwortsammlung durchgeführt. Hierbei sollten Arbeiter explizit annotieren, ob eine Antwort aus dem Text ablesbar ist oder durch Hintergrundwissen inferiert werden muss. Außerdem wurde den Arbeitern die Möglichkeit gegeben, Fragen als nicht beantwortbar zurückzuweisen.

*Experimente.* Im Rahmen dieser Arbeit wurden verschiedene Baseline-Systeme implementiert und trainiert, um die Schwierigkeit des Corpus' abzuschätzen. Außerdem wurde MCScript von uns für einen Shared Task im Rahmen der SemEval-Workshopreihe benutzt, bei dem den Teilnehmern auch Ressourcen für Skriptwissen zur Verfügung gestellt wurden.

Die Resultate unserer Experimente und des Shared Tasks zeigen, dass die Fragen in MCScript besonders für die Baseline-Modelle schwierig zu beantworten sind und dass der Datensatz generell anspruchsvoll ist. Allerdings kommen wir auch zu unerwarteten Erkenntnissen: Erstens finden wir, dass Skriptwissen nicht nötig ist, um auf den Daten eine gute Performanz zu erreichen. Zweitens ergeben unsere Untersuchungen, dass Skript-basierte Fragen nicht schwieriger zu beantworten sind als andere Fragen.

Basierend auf der Analyse der Resultate eines der am besten abschneidenden Systeme des Shared Tasks identifizieren wir zwei Gründe für diese unerwarteten Effekte: Erstens enthalten die Daten eine größere Anzahl an Fragen, die unabhängig von einem Text immer dieselbe Antwort haben. Dies ist vermutlich darauf zurückzuführen, dass Arbeiter die Texte nicht gesehen haben und Fragen nur auf Basis eines Szenarios formuliert wurden. Zweitens gibt es eine bestimme Anzahl an Fragen, die nach sehr generellen Informationen über ein Szena-

rio fragen, was wiederum Inferenz über eine generelle Art von Allgemeinwissen erfordert, welches nicht von Skripten abgedeckt wird.

*MCScript2.0.* Basierend auf diesen Erkenntnissen haben wir eine Revision der Fragenerzeugung vorgenommen und ein neues Corpus erstellt, *MCScript2.0*. In dem Experiment zur Frageerzeugung wurde Arbeitern dieses Mal ein konkreter Text gezeigt. Arbeiter wurden dann dazu angehalten, Fragen zu markierten Nominal- und Verbalphrasen zu formulieren. Der komplette Satz, der die Phrasen beinhaltet, wurde danach aus dem Text gelöscht. Arbeiter, die sukzessive Antworten formulieren sollten, waren dadurch gezwungen, die entsprechende Information aus Hintergrundwissen zu inferieren, weil sie diese nicht mehr aus dem Text entnehmen konnten.

MCScript2.0 umfasst mehr als 3.400 Texte bei mehr als 19.000 Fragen. Die Anzahl an Skript-basierten Fragen in MCScript.2.0 ist im Vergleich zu MCScript signifikant größer, bei über 50%. Mit mehreren Modellen schätzen wir die Schwierigkeit von MCScript2.0 ab, unter anderem mit dem Gewinnermodell des Shared Task. Wir zeigen, dass Skript-basierte Fragen nun für Modelle ohne Zugriff auf Skriptwissen schwieriger zu beantworten sind. Außerdem zeigen wir, dass selbst das Top-Modell, welches einen Wissensgraphen für Allgemeinwissen (ohne Skriptwissen) verwendet, systematische Schwierigkeiten hat. Dies legt nahe, dass eine wichtige Teilklasse von Leseverstehensfragen in MCScript2.0 Skriptwissen zur Beantwortung erfordert.

*Für meinen Vater.*

# Acknowledgments

First of all, I want to thank my advisor Manfred Pinkal, who has guided me through most of the years of my academic path. Manfred provided great guidance and always had an answer, no matter how difficult the question was. Most importantly, Manfred showed me how academia works: I (hopefully) learned how to ask the right questions, how to proceed with research when you seem to be stuck, and how to professionally handle conflict situations.

Second, I want to thank Michael Roth. Michael was an outstanding advisor, who always provided great ideas and input, with his brilliant mind and broad knowledge of both theoretical and more technical topics. Moreover, he helped me at difficult times to keep going, and pushed me to do things that I would otherwise not have tackled (such as organizing two shared tasks, or playing board games against two of my advisors).

I also owe thanks to Stefan Thater for always having an open ear to discuss any ideas that came up, no matter if they fitted my current research or not, and for very productive brainstorming sessions.

To Alexander Koller I owe thanks because he provided invaluable advise during the second phase of my PhD. Most importantly, Alexander sparked my interest for deep learning, which is going to be a most valuable skill in the future for sure.

A big thanks goes to Andrea Horbach, who got me interested in academic research in the first place and who morphed from a great advisor and colleague to a friend throughout the years.

To Lilian Wanzare, Ashutosh Modi, Annemarie Friedrich and Carina Silberer, thanks for making my work a lot more enjoyable and for discussing not only research-related topics, but also mundane things like American politics, the horrible German bureaucracy or the best Indian restaurants in Saarland. A big "Thank You" to Diana Steffen, who helped with all small and big bureaucratic obstacles and who was always there for a small chitchat at the coffee machine.

I was very lucky to advise Hannah Seitz during the research leading to her Master's thesis. I learned a lot by supervising her, and besides, Hannah was a great colleague to work (and go to mensa) with.

I want to thank all student assistants that conducted annotations, implemented ideas that I didn't find the time for, or helped with all big and small tasks that popped up: Tatjana Anikina, Sophie Henning, Sarah Mameche, Stefan Grünewald, Damyana Gateva, Georg Seiler, Florian Pusse, Leonie Harter, Christine Schäfer and David Meier.

Doing a PhD is a lot of work, which can only be tackled with a certain degree of enthusiasm and the willingness to dedicate large amounts of time into the project. One important fact I learned during the last years is that it is not only this mindset that plays a crucial role for successfully achieving a PhD degree, but also the ability to actively take breaks from research and focus on other things to regenerate the mind. I owe lots of thanks to my band, who made it possible for me to channel my thoughts on music and to forget work from time to time.

Last and most importantly I want to thank my mother and my girlfriend for being there and providing the emotional support without which I'd never have submitted this thesis.

# Contents


**I Introduction and Background** 1

**1 Introduction** 2

1.1 Script Knowledge . . . . . 5

1.1.1 Script Structure . . . . . 5

1.1.2 Scripts and Frames . . . . . 8

1.2 Making Scripts Accessible for NLP . . . . . 10

1.3 Possible Impact of Script Knowledge . . . . . 15

1.4 Thesis Structure . . . . . 18

1.5 Contributions of this Thesis . . . . . 20

**2 Script Knowledge Representations** 24

2.1 Scripts as Paraphrase Sets . . . . . 27

2.2 Scripts as Narrative Chains . . . . . 29

2.3 Neural Script Representations . . . . . 31

2.4 Differences and Commonalities . . . . . 32

2.5 Other Commonsense Knowledge Representations . . . . . 33

II Identifying Scripts in Natural Texts 37

3 Script Parsing - Background 38

3.1 Basic Idea and Motivation . . . . . 38

3.2 Script Parsing of Naturalistic Texts . . . . . 42

3.3 Script Parsing in the Scope of this Thesis . . . . . 44

3.4 Related Work . . . . . 46

4 Script Parsing Data - The InScript Corpus 48

4.1 Data Collection . . . . . 50

4.1.1 Collection via Amazon Mechanical Turk . . . . . 50

4.1.2 Data Statistics . . . . . 51

4.2 Annotation . . . . . 53

4.2.1 Annotation Schema . . . . . 53

4.2.2 Development of the Schema . . . . . 58

4.2.3 First Annotation Phase . . . . . 59

4.2.4 Modification of the Schema . . . . . 60

4.2.5 Special Cases . . . . . 60

4.3 Data Analysis . . . . . 62

4.3.1 Inter-Annotator Agreement . . . . . 62

4.3.2 Annotated Corpus Statistics . . . . . 63

4.3.3 Comparison to the DeScript Corpus . . . . . 66

4.4 Conclusion . . . . . 67

5 Analysis of the Script Parsing Data 69

5.1 Data . . . . . 71

5.2 Lexical Entailment: Annotation Study . . . . . 72

5.2.1 Events . . . . . 73

5.2.2 Participants . . . . . 75

5.3 Clause-Level Entailment: Composition . . . . . 76

5.4 Annotation Statistics . . . 78
5.4.1 Lexical Level . . . 79
5.4.2 Clausal Level . . . 80
5.5 Conclusion . . . 81

**6 A Script Parsing Model 82**
6.1 Event Verb Identification . . . 84
6.2 Event Type Identification . . . 86
6.2.1 Baseline: Similarity-Based Model . . . 87
6.2.2 Sequence Labeling Model . . . 88
6.2.3 Alternative: Hidden Markov Model . . . 90
6.3 Experiments and Results . . . 91
6.3.1 Experimental Setup . . . 91
6.3.2 Event Verb Identification . . . 92
6.3.3 Event Type Identification . . . 93
6.3.4 Full Script Parsing Task . . . 94
6.4 Discussion . . . 95
6.4.1 Error Analysis . . . 95
6.4.2 Entailment Type and System Performance . . . 97
6.5 Summary . . . 99

**III Script Knowledge for Machine Comprehension 102**

**7 Background and Motivation 103**
7.1 Machine Comprehension and Story Understanding . . . 104
7.1.1 Early Story Understanding Systems . . . 106
7.1.2 Machine Comprehension Datasets with a Focus on Commonsense Reasoning . . . 108
7.2 Text-Based Machine Comprehension Tasks and Systems . . . 111
7.2.1 Data Sets . . . 112

7.2.2 Systems . . . 113
7.3 Other NLU Tasks for Evaluating Commonsense Inference . . . 114
7.4 Summary . . . 116

**8 MCScript: A Dataset for Assessing the Contribution of Script Knowledge for Machine Comprehension 118**
8.1 Pilot Study . . . 120
8.2 Data Collection . . . 121
8.2.1 Scenario Selection . . . 121
8.2.2 Texts . . . 121
8.2.3 Questions . . . 121
8.2.4 Answers . . . 122
8.2.5 Data Post-Processing . . . 123
8.2.6 Answer Selection and Validation . . . 123
8.3 Data Statistics and Analysis . . . 124
8.4 Conclusion . . . 128

**9 Experiments on MCScript 129**
9.1 Baseline Experiments . . . 131
9.1.1 Models . . . 131
9.1.2 Results and Evaluation . . . 133
9.2 Shared Task . . . 136
9.2.1 Participants . . . 136
9.2.2 Results . . . 140
9.2.3 Discussion . . . 142
9.3 Assessing the Contribution of Script Knowledge . . . 144
9.3.1 Logistic Model . . . 145
9.3.2 Discussion . . . 146
9.4 Conclusion . . . 150

10 MCScript2.0: A Revised Machine Comprehension Dataset 152
10.1 Corpus Creation . . . 154
10.1.1 Text Collection . . . 155
10.1.2 Question Collection . . . 155
10.1.3 Answer Collection . . . 157
10.1.4 Answer Candidate Selection . . . 158
10.2 Corpus Analysis . . . 160
10.2.1 General Statistics . . . 160
10.2.2 Questions . . . 161
10.3 Experiments . . . 162
10.3.1 Models . . . 163
10.3.2 Human Upper Bound . . . 164
10.3.3 Results . . . 164
10.4 Conclusion . . . 167

IV Future Work and Conclusion 169

11 Outlook 170

12 Conclusion 175

List of Figures 177

List of Tables 182

Appendices 202

Appendices 203
A First Version of Event and Participant Annotation for InScript . . . 203

B Final Version of Event and Participant Annotation Guidelines for InScript 211
C Lexical Entailment Annotation Guidelines - Events . . . . . 224
C.1 Task . . . . . 224
C.2 Annotation Material . . . . . 224
C.3 Guidelines for the Manual Annotation . . . . . 225
C.4 Annotation Labels . . . . . 228
C.5 Some Additional Remarks . . . . . 233
C.6 Additional Material . . . . . 235
D Lexical Entailment Annotation Guidelines - Participants . . . . . 236

# Part I

# Introduction and Background

Chapter 1

# Introduction

For ages, humanity has sought to find out what *Intelligence* is. In the first half of the twentieth century, research had proven that the human brain is basically a network that fires electric pulses between billions of neurons. Modeling such a network with a computing machine seemed interesting, since it would be the first step for enabling a computer to "think" like humans. Such ideas of an automated modeling of brain function were fueled by influential works on computation theory (Turing, 1937) and information theory (Shannon, 1948): The first steps towards modeling *Artificial Intelligence* began. At the time, these notions were merely theoretical, since there were only a few computers available; and these lacked the computing power to actually run and test neural models. Nevertheless, the interest in the topic was sparked. Two main events helped to establish the field of AI:

First, Alan Turing proposed in the 1950's, that a machine would be truly "intelligent" if it could carry a conversation with a human counterpart that was natural enough to trick the human into believing she was not talking to a computer (Turing, 1950), an idea known as the *Turing Test*. At the time of publication, the thought of having a computer that was intelligent enough to trick a human was futuristic and unachievable within decades.

Second, the Dartmouth Summer Research Project on Artificial Intelligence was held in 1956. The summer school is nowadays often seen as a starting point for modern research on AI. In the meeting, John McCarthy and colleagues formally established the research field, gave it

its name and built a first, small community of people interested in formalizing and modeling AI.

Since these early attempts on AI, the research field has emerged as one of the most important areas of computer science. Research on AI has made rapid progress, fueled by an abundance of computing resources: According to Moore's law (Moore, 1965), computing power doubles roughly every second year[1], paving the way for complex software and learning algorithms. Also, the advent of machine learning, and, more recently, deep learning and artificial neural networks, made it possible to use vast amounts of data for training intelligent systems.

Very prominently, methods of *Natural Language Processing (NLP)* have benefited from this trend: Personal assistants such as *Siri*[2] and *Google Assistant*[3], that are able to communicate with a user via natural language, are neatly integrated into our personal lives. Speech recognition is integrated into many devices that we use every day (cf. e.g. McGraw et al. (2016)). There has been a lot of progress in the area of question answering: Web search engines such as WolframAlpha[4] enable a user to ask natural language questions and retrieve answers from the web. The *Watson* AI[5], another system that is able to answer natural questions by looking up answers on the web, became famous because it was able to beat expert human competitors in the quiz show *Jeopardy*[6].

By design, such intelligent systems are highly specialized and tailored for the task at hand: Text understanding systems are good at reading a text and answering special kinds of questions based on it, but often unable to answer questions that ask for details that are trivial for humans, but that go beyond what is written in the text.

An example is shown in Figure 1.1, which shows a failed search query in WolframAlpha. The search engine is unable to find an answer to the question *Who prepared a drink for James Bond at the bar?* on the web, although this situation appears in the majority of James Bond movies, and there are movie scripts available on the web. We as humans are able to find the

[1] Actually, the law predicts that the number of transistors in a dense integrated circuit doubles every two years, which is no *guarantee* for a doubling of the computing power - but it still remains a rough estimate.

[2] https://www.apple.com/de/ios/siri/

[3] https://assistant.google.com/

[4] http://www.wolframalpha.com/

[5] https://www.ibm.com/watson/

[6] https://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html

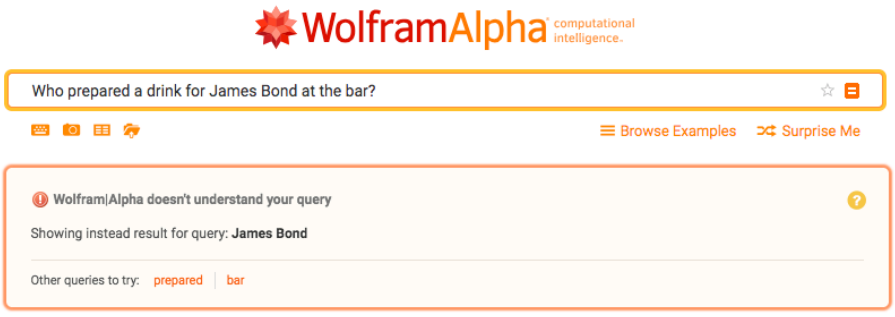



**Figure 1.1:** A failed search query in WolframAlpha.

correct answer *the bartender*, since we have access to *Commonsense Knowledge* (van Holthoon and Olson, 1987). It is defined as "the basic level of practical knowledge and judgment that we all need to help us live in a reasonable and safe way" [7] and contains the information that drinks in bars are prepared by bartenders.

Although these types of inference are very natural for humans, enabling computer programs to perform such commonsense-based inferences remains a widely unsolved problem. The reasons for this are two-fold: First, the acquisition and representation of commonsense knowledge is an open problem and a topic of research. A system that is able to find an answer for the question would need a representation of the fact that bartenders prepare drinks. Second, making commonsense knowledge available for a computer system is challenging. A system does not only need a representation of the information that bartenders prepare drinks, but it also needs to associate and align this information with the question in order to make use of it.

The information that bartenders prepare drinks in bars is subsumed by *Script Knowledge*, one of the most interesting types of commonsense knowledge and a focus of this dissertation. Script knowledge is knowledge about everyday activities, such as going to a restaurant, planting flowers, ordering pizza etc. Specifically, it is knowledge about the typical steps that are performed in an activity, together with the persons and things that play a role in the activity. In the example, script knowledge about the *going to a bar* activity would subsume the fact that bartenders prepare drinks in bars.

The term *script knowledge* was first introduced by Schank and Abelson (1975) and later refined by Barr and Feigenbaum (1981), who define *Scripts* as building blocks for knowledge

[7] https://dictionary.cambridge.org/dictionary/english/common-sense

about everyday procedures, stored in the human memory. The theory of script knowledge is also grounded in cognitive psychology. Graesser et al. (1979) for example found evidence that humans process everyday activities by recalling former instances of the same activity, building script-like structures in their brain as a consequence. Adams and Worden (1986) found that script knowledge is acquired at a very young age and refined over time: With increasing age, humans get better at identifying atypical episodes in scripts. These findings strongly suggest that scripts are not learned at any one point in time, but by repeatedly being exposed to standardized situations in a natural everyday environment.

## 1.1 Script Knowledge

In this section we lay formal basics for the concept of script knowledge. A more elaborate discussion of different representations and related work can be found in Chapter 2.

### 1.1.1 Script Structure

Scripts contain commonsense knowledge about everyday activities. Since there are many different types of such activities, there are as many different types of scripts that describe them. There are several elements that make up a script:

**Scenario.** *Scenarios* are different types of everyday activities that are described by script knowledge. In this thesis, we will use SMALL CAPS for scenario names. Some examples are VISITING A RESTAURANT, CLEANING THE FLOOR, or TAKING A BUS . Scripts can describe various aspects of everyday life: Some belong to the domain of leisure activities, such as GOING ON VACATION, PLAYING TENNIS, GOING BOWLING, some describe household duties such as CLEANING THE KITCHEN, or MAKING A BED, or transportation activities, such as GOING ON A BUS or DRIVING A CAR, and so forth.

Often, there is no sharp line between scenarios, and there are scenarios that complement or overlap each other: When going on vacation, one usually takes a plane, and a taxi to the airport. The TAKING A TAXI script can thus be a part of FLYING IN A PLANE, which is a part of GOING ON A VACATION, and so forth. Several scenarios – and as a consequence several scripts – can be activated within a single narration.

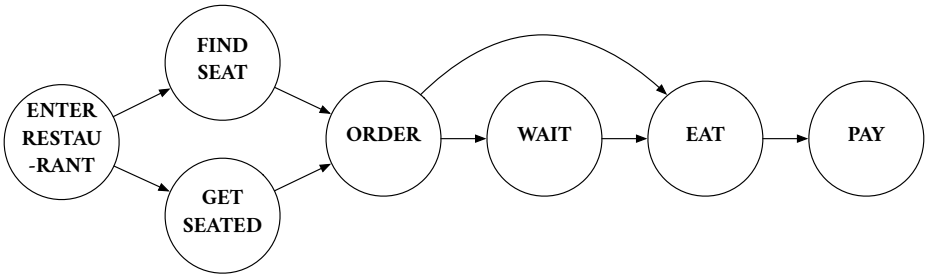



**Figure 1.2:** An example script graph for the GOING TO A RESTAURANT scenario.

**Events.** The main components of a script are *events*. Each event describes an action that takes place in a scenario, such as ORDER or EAT in the GOING TO A RESTAURANT script. Figure 1.2 shows events of the GOING TO A RESTAURANT scenario, such as the aforementioned ORDER or EAT, and other events, such as FIND A SEAT or GET SEATED. As for scenarios, we will use SMALL CAPS for event types.

In natural language, there are many ways in which events can be expressed: The EAT event for example can be referred to as in Examples 2 through 4.

(2) Andrea ate her steak.

(3) Andrea enjoyed her steak.

(4) Andrea gulped down her steak.

Although the verbs in the examples are different, they all describe the same event type within the GOING TO A RESTAURANT SCENARIO.

**Participants.** Script knowledge contains knowledge about the *participants* of a scenario. Participants are the roles of a script, which are instantiated by persons or things that play a role in the script. Examples for participants in the GOING TO A RESTAURANT script are the GUEST, the WAITER, MONEY etc. Typically, certain participants are associated with specific events. GUEST and WAITER will usually participate in the ORDER event, while the latter is normally not involved in EAT. Still, each participant type is usually active and accessible throughout the whole script. We will use SMALL CAPS also for participant types.

Like events, participants can be referred to in various ways. There are two levels of variation. First, a participant role can be filled with completely different types of entities. *Steak*, *salad* or *pasta* could all be instances of the FOOD participant within one narration. Second, the same participant can be referred to using expressions of differing granularity: Depend-

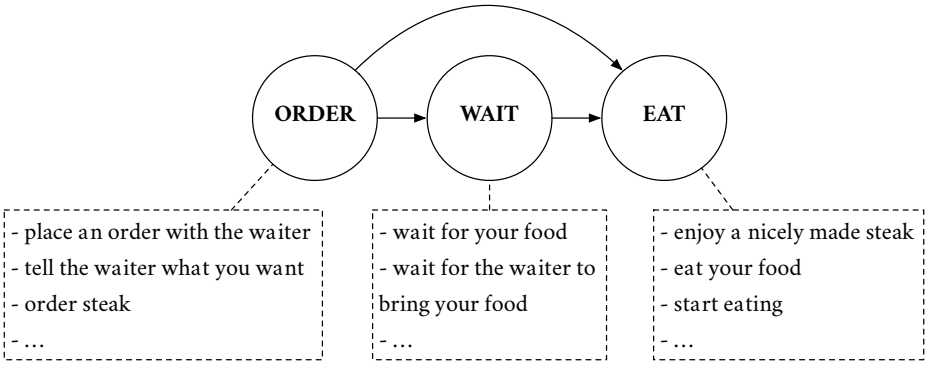



**Figure 1.3:** Representing events as paraphrase sets.

ing on the narration, the FOOD participant can be referred to as *food*, *beef*, *tasty and juicy fillet* and so forth.

**Event Structure and Temporal Order.** Script events are temporally ordered. For example, EAT typically comes after WAIT, and is followed by PAY. To illustrate this temporal ordering information, scripts can be portrayed as directed graphs, as shown in Figure 1.2. The vertices stand for script events, the directed edges show the typical temporal ordering of the events. It is assumed that scripts have a *partial ordering* (Regneri, 2013), rather than a strict temporal ordering structure. Consider the GOING TO A RESTAURANT scenario: Usually, PAY comes after EAT, but it is also possible that PAY precedes EAT, for example in a fast food restaurant. Also, there are events that are optional in a script scenario, such as GIVE A TIP, and some events can be alternatives to others, such as WALK TO RESTAURANT and DRIVE TO RESTAURANT.

While the temporal order of events will be addressed in this dissertation, there are script representations that additionally encode other relevant types of relational information not covered in this work. An example is causality: The event EAT is only plausible if the guest has *received* his or her food, and GET SEATED can only take place, if the restaurant was *entered* before. Causality is usually encoded by means of pre- and post conditions (Arnold, 2016). Preconditions are conditions that need to be fulfilled in order for an event to happen: A precondition of the EAT event is that the food was brought to the table. A postcondition defines a state of the world after an event is executed. A postcondition to EAT is for example that the food is gone.

**Representing Scripts.** There have been several efforts to represent script knowledge in computational linguistics research. In this thesis, we will follow Regneri et al. (2010) and rep-

| *Name* | Commerce_Pay |
|---|---|
| *Frame Elements* | Buyer<br>Goods<br>Money<br>Rate<br>Seller |
| *Predicates* | disburse.v, disbursement.n<br>pay.v, payer.n, payment.n<br>shell out.v |

**Figure 1.4:** The *Commerce_Pay* frame, from FrameNet (Ruppenhofer et al., 2006)

resent scripts in terms of *paraphrase sets*. Here, each script event is represented as a set of short, telegram-style phrases that describe the event in simple words. Figure 1.3 shows paraphrase sets for some events in the GOING TO A RESTAURANT script. Parallel to events, participant types are represented as sets of noun phrases describing the respective type. Such paraphrase sets provide important linguistic realization variants of both participants and events, which can be utilized in applications that require script-based inference. We will elaborate on paraphrase sets and other script representations in Chapter 2.

### 1.1.2 Scripts and Frames

The theory of *Frame Semantics* (Fillmore, 1982) is related to script knowledge. The frame semantic theory assumes that a single word cannot be understood without knowing about semantic concepts that are related to the word. All the concepts that are required to understand the full meaning of a word are organized in so-called *frames*.

An example is the verb *pay*. Frame semantics defines that in order to understand the meaning of the word *pay*, one needs the knowledge that there is for example an object that is paid for, a person that pays, and a means of payment. These concepts, also referred to as *roles*, are part of the frame that is associated with the verb *pay*. Figure 1.4 shows the frame that is associated with *pay*, *Commerce_Pay*, taken from *FrameNet* (Ruppenhofer et al., 2006), a large database of common frames for English. The example lists all roles that are associated with

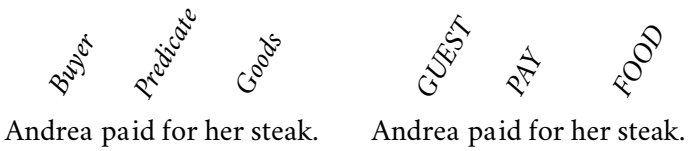



**Figure 1.5:** Frame analysis (left) and Script analysis (right).

the frame (referred to as *Frame Elements* in FrameNet): In the *Commerce_Pay* frame, there is for example a *Goods* role, i.e. the thing that is bought by the *Buyer* from the *Seller*.

FrameNet also lists predicates, which are frame-evoking elements, i.e. the verbs or nouns that activate a frame. In the example, verbs like *pay* or *disburse* are listed as words that evoke the *Commerce_Pay* frame. This information can be used to identify the most prominent frame-evoking elements in a text.

Since frames usually describe single actions, they are similar to script events: A verb that evokes the *Commerce_Pay* frame can instantiate the PAY event of the GOING TO A RESTAURANT script, as shown in Figure 1.5. The nouns that fill the frame roles correspond to instances of script participants: *Andrea* fills the *Buyer* role in the frame analysis, and it instantiates the GUEST script participant. In general, an action that expresses a script event evokes a frame at the same time. Parallel to events, an entity in a narration about some everyday activity usually instantiates a script participant and fills a frame role.

However, scripts and frames are different in some important aspects:

First, script events are always specific to a scenario, while frames are defined irrespective of a text genre, domain or topic. Figure 1.5 also illustrates this: Andrea instantiates the GUEST participant, but one could also imagine that the sentence takes place in a narration about GOING TO THE GROCERY STORE, e.g. a situation in which she pays at the grocery store counter. In this case, *Andrea* would not instantiate the GUEST participant, but the CUSTOMER participant. As for the frame analysis, the entity fills the *Buyer* slot for the *Commerce_Pay* frame in both cases.

Second, a central aspect of script knowledge is information about the temporal order of events, which is not encoded in frames: A script describes a whole chain of actions, i.e. it is a discourse-level model, whereas a frame only describes a single action. Frames can in principle also be organized in a way that goes beyond single actions: In FrameNet for exam-

ple, frames are linked with different relations (such as *inheritance*, a subsumption relation, or *using*, where one frame is a presupposition to the other). FrameNet especially also defines so-called *scenario frames*, which describe more complex actions that consist of several subframes. In some respects, such scenario frames are similar to scripts, since they provide a more structured representation of frame actions that typically occur together. However, there are not many such frames available in the FrameNet database.

Due to the interrelations between frame and script knowledge, frame knowledge can provide important information for identifying script events and participants, an aspect that we will more closely look at in Chapter 6.

## 1.2 Making Scripts Accessible for NLP

(5) Person A: "Yesterday, I went to this new Argentinian restaurant to have dinner. I enjoyed a tasty steak."
Person B: "What did you pay?"

Script knowledge enables everyday communication between people and makes it highly efficient. In Example 5, the question uttered by person B makes perfect sense, even though person A never mentioned that he or she paid. While left unmentioned, person B can infer that A ordered and received food, and that she ate and paid before leaving. Additionally, B should know that a waiter probably brought the food, which was prepared by a chef. A assumes a "common ground" (Stalnaker, 2002) with the listener, which in this case contains script information about the GOING TO A RESTAURANT SCENARIO. Upon hearing the first utterance, B will activate a mental representation of the GOING TO A RESTAURANT script, which makes the aforementioned information accessible.

The fact that this information is left unmentioned can be explained with the Gricean conversational Maxim of Quantity (Grice, 1975), which states that a speaker tries to be as informative as possible, but just as informative as necessary. In this case, A tries to avoid providing unnecessary details that are already known to B.

**Script Acquisition** While inter-human communication benefits from the fact that information based on script knowledge is often left implicit, this poses a challenge for the auto-

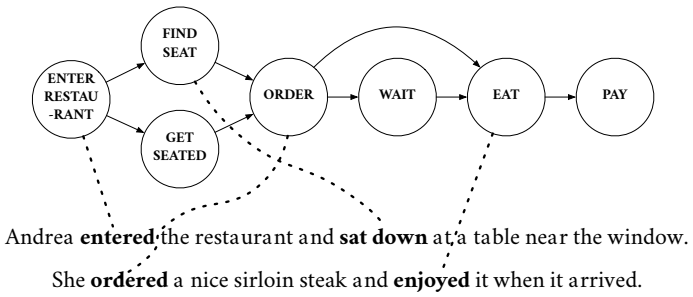



**Figure 1.6:** An example for script parsing.

matic acquisition of script knowledge on a large scale: Extracting script knowledge from natural texts is difficult, since many events are mentioned very infrequently. One possibility to circumvent this difficulty is by asking crowd workers to write down explicitly which steps are needed to conduct a given everyday activity (cf. Chapter 2 for details). Based on these crowdsourced data, scripts can be represented as paraphrase sets, as described above.

**Script Parsing** A fundamental prerequisite to utilize script knowledge for natural language understanding and inference on text, is to align the events and participants of a script with the text that is to be processed, a task also known as *script parsing*, *text-to-script mapping* or *text-to-script alignment*.

Figure 1.6 illustrates the alignment of script events with a short narrative text. The dotted lines indicate the mapping of events to their instantiations in the restaurant narration. If a system is able to align *enjoyed* with the EAT event, it can draw inferences based on script knowledge on the text and model expectations about what comes next. It can also infer that the WAIT event took place, even if it was not mentioned.

Script parsing is a non-trivial task, and an alignment model needs to be able to cover different problems and challenges. First, as mentioned in the last section, there are many different ways in which the same event can be instantiated. A system that tries to align a script with a text should be able to recognize that each of *eat*, *enjoy*, and *gulp down* are used to express the same type of event. This is especially challenging because of words such as *enjoy*, which is no paraphrase of *eat* outside of the script context.

Also, events can be described in both a more and a less explicit way. Example 6 shows how the EAT event can be instantiated very explicitly, by mentioning the movement of the fork

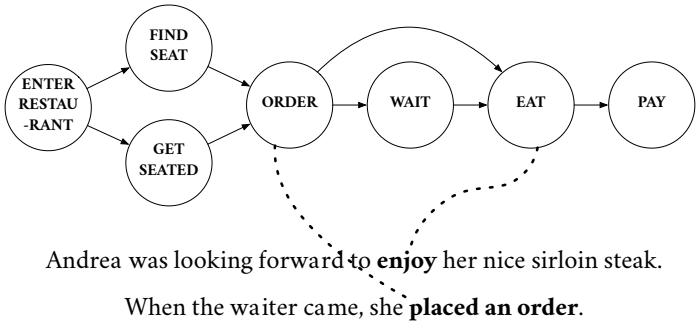



**Figure 1.7:** The narrative order of events can deviate from the order in which they actually took place.

to the mouth and chewing. Both verbs together describe the EAT event.

(6) Andrea moved the fork to her mouth and chewed thoroughly.

Second, another key challenge in the automated alignment of scripts with text is the fact that the narrative order of events doesn't necessarily correspond to the order in which the events actually take place. Figure 1.7 gives an example for this. Although the EAT event is mentioned first, it takes place only after ORDER. A model that leverages script information needs to abstract away from the textual order in this case.

In some cases, the order in which two events can happen is naturally exchangeable. Examples 7 and 8 are both plausible:

(7) Andrea checked the menu to decide on a main course. She then enjoyed a steak.

(8) After eating her steak, Andrea checked the menu again to choose a dessert.

The event CHECK THE MENU can happen both before or after EAT, with both possibilities being plausible. A model that assumes a strict ordering of events will struggle with such cases.

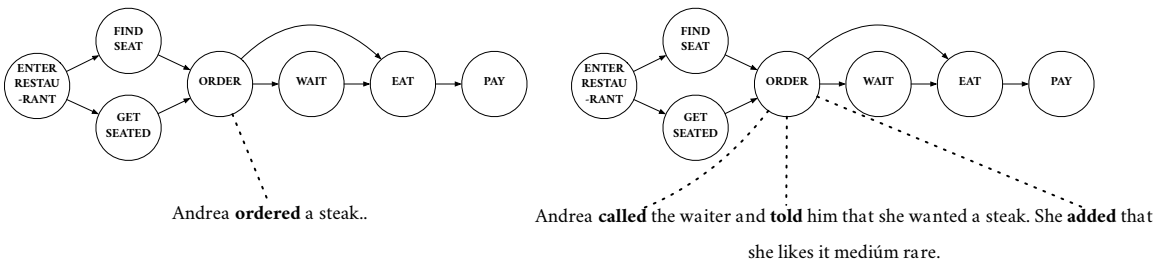



**Figure 1.8:** The granularity of event mentions can differ a lot.

| | |
|---|---|
| **I went to a restaurant** with my friend. We each ordered something we liked. The food took forever to arrive. When it did arrive, the waiter apologized for the wait. | **A customer ordered a cheesesteak from a waiter with American cheese.** The waiter forget to tell the cook to use American instead of cheddar. When the waiter realized the mistake it was too late. |

**Figure 1.9:** Two example stories form the ROC stories dataset (Mostafaza deh et al., 2016).

Third, events can be instantiated in differing granularity. The sentences in Figure 1.8 both describe the ORDER event. While the first one consists just of one word, the second mention of the event is split up into several steps: *calling*, *telling* and *adding*. The problem here is that the granularity of instantiations of the same event may differ, i.e. several smaller steps can make up one event in a text. A script representation tries to simplify, by just assuming that there is only one ORDER event. Such cases are difficult to process: The granularity of event expressions in a text can differ from the event representation of the script.

In many cases, it is also difficult to tell if a verb in a text instantiates an event of a given script at all. In Examples 9 and 10, the verb *enjoy* is used in different contexts. In the first case, it describes an instance of the EAT event. In the second case, the verb does not describe a script event, since *enjoying the weather* is not a part of the GOING TO A RESTAURANT script. For a model, this case is critical, since the exact same verb is used.

(9) Andrea went to the restaurant. She sat down and enjoyed a nice steak.

(10) Andrea went to the restaurant. She sat down and enjoyed the nice weather.

Aside from difficulties in recognizing events, there are some more general challenges for leveraging script knowledge for language processing that go beyond the scope of this thesis and that will be mentioned only briefly here.

For example, there are many ways in which script knowledge can be activated or evoked. Figure 1.9 shows the beginning of two example stories from the ROC stories data set (Mostafaza deh et al., 2016). This data set contains 100,000 crowdsourced narrative stories of 5 sentences each that talk about arbitrary everyday activities. Both stories in the example talk about a restaurant visit, so the GOING TO A RESTAURANT script is activated. In the left story, this happens very explicitly, using the phrase *I went to a restaurant*. In the second case however,

Last night Michael and I went to the lovely Dr Meg's for dinner . (...) As always when invited to dinner or lunch or just anywhere that I have the opportunity I stuck my hand up volunteered dessert , another chance to test the new oven . I tried out this upside down cake from Bill Grangers, Simply Bill. As I have mentioned before , I love plums am always trying out new recipes featuring them when they are in season . I didn't read the recipe properly so was surprised when I came to make it that it was actually cooked much in the same way as a tarte tartin , ie making a caramel with the fruit in a frying pan first , then pouring over the cake mixture baking in the fry pan in the oven before turning out onto a serving plate , the difference being that it was a cake mixture not pastry . (...) As I made this very early in the morning when Miss Chloe was sleeping I couldn't serve hot out of the oven as suggested , instead serving at room temp with some double cream . Everyone had 2 big helpings which I think was a pretty good indication that it was a hit , the recipe is definitely a repeater. As always, Bill didn't let me down with this cake , soft cooked plums , sugary caramel on the top with a beautiful moist cake underneath mmm of course I woke up wishing I hadn't left the rest of the cake with Megan , though my body is thanking me . (...)

**Figure 1.10:** An example story from the Spinn3r data set (Burton et al., 2009) that talks about the HAVING DINNER scenario (in blue), as well as BAKING A CAKE (in green).

the restaurant is not mentioned once, but the story starts right away with a mention of the script event ORDER. A text understanding model that tries to access script knowledge needs to be able to recognize both kinds of script evocation.

Another characteristic of script knowledge that makes it difficult to use in language processing is the fact that in most narrations, several scripts are intertwined. Figure 1.10 shows a fragment of a story from the Spinn3r blog story corpus (Burton et al., 2009), a large collection of blog posts crawled form the web. As can be seen, the story starts out with a HAVING DINNER scenario (in blue), deviating to a narration about BAKING A CAKE (in green), only to get back to HAVING DINNER. The fact that often, several scripts are activated throughout a text, makes the automated alignment of scripts with the text difficult. A more elaborate example can be found in Chapter 3.

## 1.3 Possible Impact of Script Knowledge

Script knowledge is a fundamental prerequisite for human natural language understanding. Consequently, there are also numerous NLP problems and applications that would greatly benefit from the availability of script knowledge.

### Question Answering

Text understanding models should in general benefit a lot from the incorporation of background knowledge in the form of scripts. One very prominent application for text understanding models is question answering (QA). The idea is simple: Given a text or a database, a system has to find the correct answer to a question. In recent years, QA has become a prominent application in NLP because of the availability of the aforementioned personal assistants, such as *Siri* or *Google Assistant*, whose main purpose is to answer questions posed by a user. There are various types of QA tasks: For example open QA, in which the answer needs to be found from a large database, ontology, or the web, and reading comprehension - based QA (also referred to as *machine comprehension*), in which a system needs to read a text and find the answer in the text[8].

Most current QA systems utilize complex architectures for reading the text, and finding connections between text/knowledge base, question and answers. However, there are only few attempts to incorporate external knowledge. One notable exception is Yang and Mitchell (2017), who use WordNet[9] (Miller, 1995) and NELL[10] (Carlson et al., 2010; Mitchell et al., 2015, 2018) to embed knowledge graph triples and enhance the text representation with them. The knowledge that is used here is purely semantic (WordNet) or factual (NELL).

(11) It was a long day at work and I decided to stop at the gym before going home. I ran on the treadmill and lifted some weights. Once I was done working out, I went in the locker room and stripped down and wrapped myself in a towel. I went into the

[8] For a nice overview s. Hirschman and Gaizauskas (2001) or http://matt-gardner.github.io/paper-thoughts/2016/12/08/reading-comprehension-survey.html

[9] https://wordnet.princeton.edu/

[10] http://rtw.ml.cmu.edu/rtw/kbbrowser/

sauna and turned on the heat. I let it get nice and steamy. I sat down and relaxed. I stayed in there for only about ten minutes because it was so hot and steamy.

**Q1:** Where did they sit inside the sauna?

**a.** On the floor. **b.** On a bench.

A multiple-choice QA example from *MCScript* (s. Chapter 8), a reading comprehension corpus with a focus on commonsense reasoning, that might benefit from the application of script knowledge, is given in Example 11. Here, a human can find the answer to the question easily: Usually, people sit on benches inside a sauna, and not on the floor. This information is subsumed by script knowledge about GOING TO A SAUNA, but not mentioned in the text. Question answering and, more specifically, machine comprehension are a central focus of this dissertation (s. Part III of the thesis).

### Recognizing Textual Entailment

Textual Entailment is defined as a directed relation between a pair of statements, usually called *text* (t) and *hypothesis* (h): "t entails h if, typically, a human reading t would infer that h is most likely true." (Dagan et al., 2006) The task of Recognizing Textual Entailment (RTE) is to decide, given a text-hypothesis pair, whether the text entails the hypothesis, contradicts it, or whether they are independent of each other.[11]. Similar to QA, RTE is a straightforward application for text understanding models and thus an interesting area for the application of script knowledge.

The Stanford Natural Language Inference data (Bowman et al., 2015) is the largest dataset for RTE. An example from the data set for an entailment pair that shows how RTE could benefit from script knowledge from these data is given in 12:

(12) **Text:** A man wearing a blue and white plaid shirt is sitting at an outdoor restaurant table, looking at a menu while a waiter waits to take his order.

**Hypothesis:** The man is deciding what to order.

[11] Note that there are also multiple other flavours of RTE with differing degrees of entailment relations: Partial entailment, reverse entailment, etc.

ORDER is not contained in the text, but a valid event in the GOING TO A RESTAURANT scenario. Also, script knowledge would imply that LOOK AT A MENU is usually followed temporally by DECIDE WHAT TO ORDER, which could help in recognizing the entailment between text and hypothesis.

## Coreference Resolution

A coreference resolution system tries to identify all expressions in a text that refer to the same entity. Script knowledge provides important information that can be leveraged for coreference resolution.

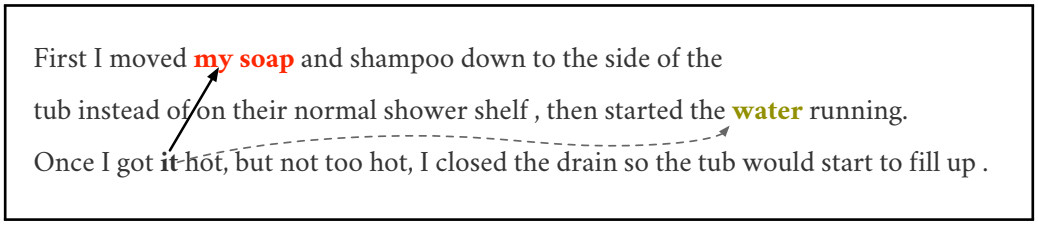



**Figure 1.11:** Script knowledge can improve coreference resolution systems.

The text in Figure 1.11 is taken from the InScript corpus (s. Chapter 4) and illustrates the idea. When running the Stanford CoreNLP coreference resolution system[12] on the two sentences, the system identifies *my soap* as antecedent of *it*, which is not correct. If the model had access to script knowledge, this would help to find that in the TAKING A BATH script, the *water* is usually heated up rather than the *soap*, which would in turn help to find the correct antecedent.

Script knowledge has been successfully used to predict upcoming discourse referents, a task that is highly related to coreference resolution. Modi et al. (2017) show that a discourse referent prediction model can be improved with script-based features.

[12] https://corenlp.run/, CoreNLP version 3.9.2, as tested on Jun 17, 2019

### Text Generation

I decided to go to the grocery store and get in my car and drive to the shop and get a list of things that I need to buy. I put my groceries in the cart, and then to took my cart to UNK where I would get the items on the list.

**Figure 1.12:** An example generated story from Shkadzko (2017).

In the area of narrative story generation, script knowledge has a major impact. Based on script data, coherent narrative texts can be generated. Figure 1.12 shows a narrative story from Shkadzko (2017) that is generated based on script data only.

The example illustrates that the task is feasible, but it also shows that there is space for improvement. There is a small inconsistency in the plot: The shopping cart is mentioned, although it was never taken from the parking spot. Causal script information would help here, a shopping cart needs to be taken before you can put the bought stuff into it.

## 1.4 Thesis Structure

In this section, we provide an overview of the structure of this dissertation.

**Part I: Introduction and Background**

In Part I of this thesis we provide basic background for the topics covered in this thesis. In Chapter 1, we give a general introduction and motivation for the idea of aligning scripts with narrative texts and highlight some of the challenges. We also cover NLP applications that could benefit from the inclusion of script knowledge.

In Chapter 2, we present recent efforts on learning and representing script knowledge. We present models that represent scripts as paraphrase sets and some alternatives to this representation. We discuss differences and commonalities of the representations and give a brief summary of efforts for representing other kinds of commonsense knowledge.

**Part II: Identifying Scripts in Natural Texts**

Part II of this thesis presents data collection and annotations efforts, as well as models for the

task of mapping scripts to texts, which is a central prerequisite for making script knowledge available in NLP applications and a pivotal topic of this dissertation.

In Chapter 3, we first introduce and motivate the task of *script parsing*, in which script events are aligned with event mentions in narrative texts. We provide an overview of the single sub-tasks that are part of script parsing, investigate potential challenges and give a short overview of related work.

In Chapter 4, we present the *InScript* corpus, a collection of 910 narrative texts with a full annotation of script event and participant types. This corpus serves as a gold standard for our script parsing models and annotations and provides an unique opportunity for investigating script knowledge instantiations in narrative texts.

Chapter 5 describes an annotation project that was conducted on pairs of event mentions in InScript and the corresponding script events in the *DeScript* corpus, a resource of script knowledge in the form of event paraphrase sets. We annotated each pair of event mention and corresponding event paraphrase set with an entailment label, indicating the kind of semantic inferences that needs to be modeled to align the mention with the event. this annotation gives an insight on the types of semantic relations that need to be modeled in script parsing.

Our script parsing model is described in Chapter 6. Together with similarity-based baselines, we present a conditional random field model which leverages temporal ordering information about script events for aligning event mentions in narrative texts with event paraphrase clusters. We conduct a thorough analysis of the results and discuss possible problems.

**Part III: Script Knowledge for Text Understanding**

In Part III of this thesis, we shift the focus to an end-to-end application for evaluating the contribution of script knowledge: Story understanding and machine comprehension.

In Chapter 7, we give background information on the topic. Story understanding has been proposed as an evaluation for script knowledge before, and we present a review of early systems and applications in this area. We also present machine comprehension corpora with a focus on evaluating other types of commonsense knowledge inferences, as well as other tasks for evaluating commonsense inference. Finally, we talk about the most common machine comprehension corpora and system architectures (without a focus on commonsense-based

inference).

Chapter 8 describes the data collection effort for *MCScript*, a multiple-choice machine comprehension corpus with a focus on challenging inferences based on commonsense knowledge and, more specifically, script knowledge. We describe a novel question collection method resulting in challenging questions and provide an exhaustive data analysis.

Experiments on the data are described in Chapter 9. We provide several benchmark models and present the results of a shared task that was conducted using MCScript. One of the central findings is that commonsense knowledge is not necessarily required to perform well on MCScript, and that the possible contribution of script knowledge especially is restricted.

We thus revised the data collection process, in particular the question collection, which is described in Chapter 10. We present *MCScript2.0*, a revision of MCScript focused on script events and participants that are mentioned in the reading texts. We show that benchmark models as well as a state-of-the art system that makes use of ConceptNet (Speer et al., 2017) struggle on the new data, in particular on questions that require script-based inference. This implies that the data provide more challenging test cases that require models to make use of script knowledge.

**Part IV: Future Work and Conclusions**

Part IV concludes the thesis. In Chapter 11, we talk about possible improvements of our script parsing models and discuss ways for incorporating script knowledge into machine comprehension models. In Chapter 12 we recapitulate the basic contributions of this thesis.

## 1.5 Contributions of this Thesis

This thesis investigates how script knowledge is instantiated in narrative texts by providing a data collection of script-instantiating texts, as well as annotations on the data. To the same end, this thesis presents the first scalable script parsing model. Furthermore, we provide two corpora for the end-to-end evaluation of script knowledge in a machine comprehension setting, as well as benchmark experiments on the data. The detailed contributions are as follows:

1. The InScript corpus, the first collection of narrative texts that is fully annotated with script event and participant type labels; joint work with Ashutosh Modi, Tatjana Anikina Stefan Thater and Manfred Pinkal. The author of this dissertation contributed especially during the second annotation round, where he modified and adapted the annotation schemas and supervised the annotation procedure. He also contributed significantly to write the paper and conducted additional analyses. Results of this work have been published in Modi et al. (2016); parts of that paper have been taken over literally into this thesis.

2. An entailment type annotation on pairs of event mentions in InScript and their respective paraphrase clusters in DeScript, which illustrates the importance of large script collections for script parsing; joint work with Hannah Seitz, Stefan Thater and Manfred Pinkal. The author of this dissertation advised Seitz in performing and planning the annotation and conducted annotation revisions as well as additional experiments. Results of this work have been published in Ostermann et al. (2018c).

3. The first scalable script parsing model based on a conditional random field which successfully leverages temporal ordering information about script event sequences; joint work with Michael Roth, Stefan Thater and Manfred Pinkal. Results of this work have been published in Ostermann et al. (2017).

4. The MCScript corpus, the first resource for an end-to end evaluation of script knowledge in a machine comprehension setting, as well as a range of benchmark models; joint work with Ashutosh Modi, Michael Roth, Stefan Thater and Manfred Pinkal. Results of this work have been published in Ostermann et al. (2018a).

5. A report on the results of a shared task that was conducted on the MCScript corpus, as well as a discussion about possible reasons for the restricted effectiveness of script knowledge for machine comprehension models employed in the task; joint work with Michael Roth, Ashutosh Modi, Stefan Thater and Manfred Pinkal. Results of this work have been published in Ostermann et al. (2018b).

6. The MCScript2.0 corpus, a revision of MCScript that eliminates the aforementioned restrictions, as well as experiments on the data which illustrate the complexity of the

data and the need for more sophisticated models based on script knowledge; joint work with Michael Roth and Manfred Pinkal. Results of this work have been published in Ostermann et al. (2019).

The research leading to the aforementioned contributions 3-6 was mainly conducted by the author of this thesis, from implementation and evaluation up to the design of annotations and data collection efforts. Annotations and data collections were conducted by student asisstants and crowdsourcing workers. The systems investigated for contribution 5 were implemented by participants of the shared task.

**Relevant Publications**

The following publications report on parts of the research conducted in the scope of this dissertation project.

1. Ashutosh Modi, Tatjana Anikina, Simon Ostermann, and Manfred Pinkal. InScript: Narrative texts annotated with script information. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), pages 3485–3493, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA)

2. Simon Ostermann, Hannah Seitz, Stefan Thater, and Manfred Pinkal. Mapping Texts to Scripts: An Entailment Study. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018), Miyazaki, Japan, May 2018c. European Languages Resources Association (ELRA)

3. Simon Ostermann, Michael Roth, Stefan Thater, and Manfred Pinkal. Aligning Script Events with Narrative Texts. In Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), pages 128–134, Vancouver, Canada, August 2017. Association for Computational Linguistics

4. Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. MCScript: A Novel Dataset for Assessing Machine Comprehension Using Script Knowledge. In Proceedings of the Eleventh International Conference on Language Resources

and Evaluation (LREC-2018), Miyazaki, Japan, May 2018a. European Languages Resources Association (ELRA)

5. Simon Ostermann, Michael Roth, Ashutosh Modi, Stefan Thater, and Manfred Pinkal. SemEval-2018 Task 11: Machine Comprehension Using Commonsense Knowledge. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 747–757, New Orleans, Louisiana, June 2018b. Association for Computational Linguistics

6. Simon Ostermann, Michael Roth, and Manfred Pinkal. MCScript2.0: A Machine Comprehension Corpus Focused on Script Events and Participants. In Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019), pages 103–117, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics

Chapter 2

# Script Knowledge Representations

In this Chapter, we explain the basics for the computational representation of script knowledge. Script knowledge has been of interest in computational linguistics for a long time. Early script representations were based on handwritten rules for a number of test cases and test scenarios. An example is the script knowledge representation used by the Script Applier Mechanism (SAM) (Cullingford, 1978), an early computer program. SAM used script knowledge in the form of hard-coded, manually created rules and mapped this script representation to a simple, hand-constructed text, in order to answer questions on the text. In this spirit, there were successors to SAM that all used similar script knowledge representations that were manually created and small in size (Dyer, 1982; Miikkulainen, 1993).

Recent years have seen a growing interest in representing script knowledge on a larger scale, which makes scripts accessible for applications in *Natural Language Understanding* (NLU) for the first time. Such script representations facilitate script parsing on a larger scale, which plays an important role in this dissertation. Script parsing in turn is an important prerequisite for leveraging the script information for NLU. In Chapter 6 of this thesis, we will use one of the presented script representations as a basis for script parsing.

In the next sections, we introduce the most common representations: We first look at scripts that are represented as partially ordered paraphrase sets, (semi-)automatically induced from manually crowdsourced data (Section 2.1). Second, we introduce narrative chains, a repre-

sentation that is automatically learned from large text collections (Section 2.2). Third, we introduce work on neural script representations that have become more popular in recent years (Section 2.3).

We present different approaches to induce script representations, show how the script representations can be evaluated, and which differences and commonalities between the different representations exist (Section 2.4). Script knowledge is a special kind of commonsense knowledge. More general commonsense knowledge is usually encoded in large knowledge bases, which have been used in NLU research in recent years. In Section 2.5, we look at such more general commonsense knowledge representations and introduce the most commonly used databases.

Table 2.1 gives an overview of recent work on representing script knowledge. The table contains references to the publications, the underlying kind of data, some keywords to describe the model and the evaluation setting that is looked at.

| | Paper | Data | Method | Evaluation |
|---|---|---|---|---|
| Paraphrase Clusters | Regneri et al. (2010) | Crowdsourced | Multiple-Sequence Alignment | O, P |
| | Frermann et al. (2014) | Crowdsourced | Bayesian Learning | O, P |
| | Wanzare et al. (2017) | Crowdsourced | Semi-Supervised Clustering | O, P |
| Narrative Chains | Chambers and Jurafsky (2008) | News | PMI | NC, O |
| | Jans et al. (2012) | News | n-Gram Probabilities | NC |
| | Pichotta and Mooney (2014) | News | n-Gram Probabilities + Multi-Argument Events | NC |
| | Rudinger et al. (2015b) | News | Log-Bilinear Language Model | NC |
| | Rudinger et al. (2015a) | Restaurant Stories | n-Gram Probabilities | NC |
| Neural Representations | Modi and Titov (2014) | News + Crowdsourced | Event Embeddings + Ranking Function | O, P |
| | Granroth-Wilding and Clark (2016) | News | Event Embeddings | NC |
| | Pichotta and Mooney (2016) | Wikipedia | LSTMs | NC |
| | Weber et al. (2018) | News | Variational Autoencoders | NC |

**Table 2.1:** Overview of models of script knowledge. The evaluation tasks are narrative cloze/event prediction (NC), event paraphrasing (P) and event ordering (O).

## 2.1 Scripts as Paraphrase Sets

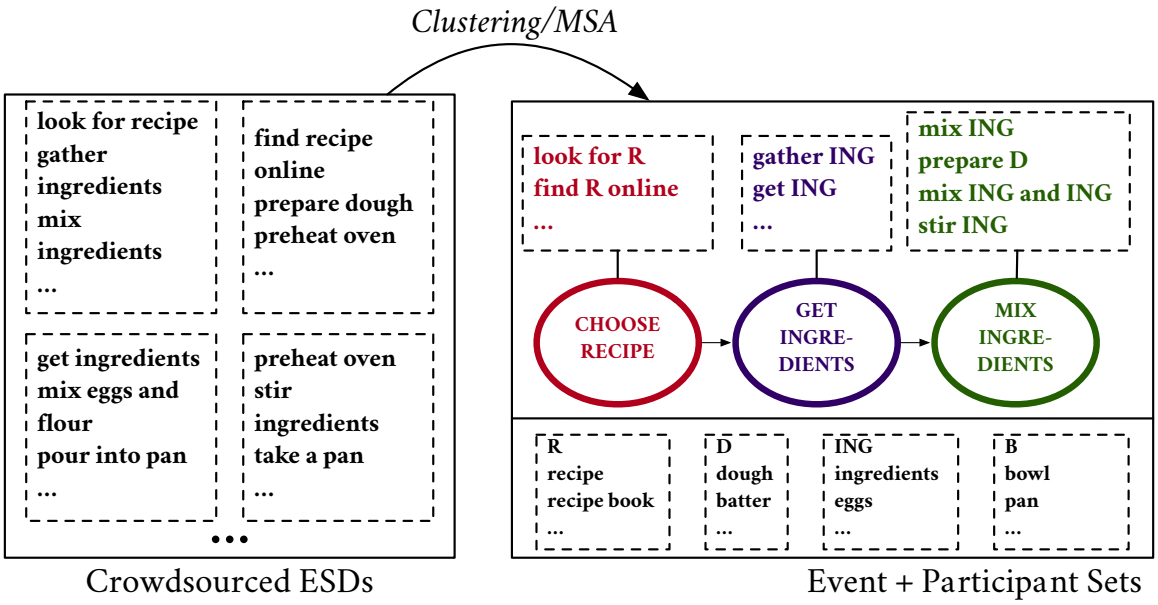



**Figure 2.1:** Representing scripts as paraphrase sets, based on crowdsourced event sequence descriptions.

**Representation.** As mentioned in Chapter 1, script knowledge comprises several types of information: Information about the events that take place in the scenario described by the script, information about the participants that play a role in the scenario, and information about the typical temporal order of events. In the representation of scripts in terms of paraphrase sets, events are represented as clusters of short event descriptions (EDs), each of which describes the event in question in telegram-style language. Likewise, participants are represented as sets of noun phrases that describe the respective participant. The right hand side of Figure 2.1 gives an example: The CHOOSE_RECIPE event is described by a cluster containing the EDs *look for R* and *find R online*. The participant cluster that describes the recipe, *R*, contains the realizations *recipe* and *recipe book*.

The basis for the (semi-)automatic induction of such script knowledge are *event sequence descriptions* (ESDs). ESDs are short, telegram-style descriptions of a single execution of a given script activity. The left side of Figure 2.1 shows 4 example ESDs. Because writing ESDs is a intuitive and straightforward task for humans, they can be acquired on a large scale via

crowdsourcing, a fast and cheap method: Existing script collections provide up to 100 ESDs per script scenario (Regneri et al., 2010; Wanzare et al., 2016). ESDs are the basis for an algorithmic induction of paraphrase clusters, as described below.

In the scope of this dissertation, we will assume an underlying representation of script knowledge in terms of such paraphrase sets.

**Models and Evaluation.** Regneri et al. (2010) use Amazon Mechanical Turk to collect 25 ESDs for 22 different everyday scenarios each. They implement a multiple sequence alignment algorithm, which originates from bio informatics, where it is used to align DNA or protein sequences. Regneri et al. (2010) use the algorithm to align several ESDs. They utilize textual similarity and align EDs *across their original ESDs* that are (1) textually similar, and that (2) appear at comparable positions in an activity. The EDs that are aligned by the algorithm form an event paraphrase cluster. Several post-processing steps are conducted subsequently in order to clean up the representation.

The typical temporal order of event clusters is read off the original ESDs: If an ED that is in cluster *a* preceded an ED that is in cluster *b* in an ESD, it is assumed that *a* precedes *b*.

To evaluate paraphrasing, Regneri et al. (2010) sample pairs of EDs from the same scenario and let MTurk workers annotate whether the two EDs describe the same event of not. For evaluating their model, they compare the annotation to the clustering computed by the MSA algorithm.

Likewise, to evaluate ordering information, they presented a sample of ED pairs to MTurk workers and let them annotate if the first one would happen before or after the second one. It can then be evaluated whether a system would predict the same order.

Frermann et al. (2014) implement a Bayesian learning approach that makes use of prior word correlation information and semantic similarity to induce a predefined number of clusters on ESDs. They employ the same evaluation method as Regneri et al. (2010).

Wanzare et al. (2017) follow a semi-supervised clustering approach to induce event paraphrase sets. They use a small number of crowdsourced seeds of EDs that are annotated to belong to the same cluster to inform an affinity propagation clustering algorithm. The algorithm makes use of both meaning representation features and positional features, which encode the typical positions of EDs inside an activity. To induce temporal order informa-

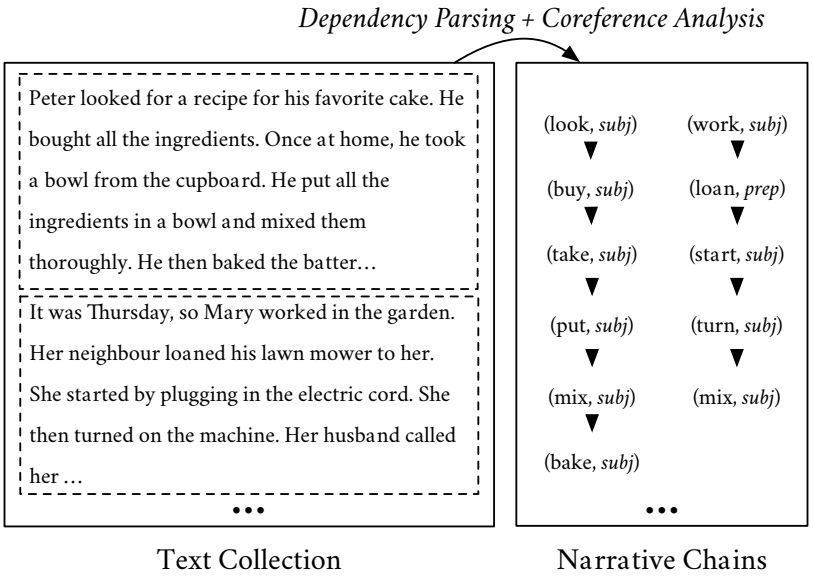



**Figure 2.2:** Representing scripts as narrative chains, based on text collections.

tion, they adopt and extend the methodology of Regneri et al. (2010).

Wanzare et al. (2017) manually create gold clusters for a subset of the data from Regneri et al. (2010) and Singh et al. (2002). This gold standard forms the basis for an even more systematic evaluation based on cluster evaluation measures such as *B-Cubed* (Bagga and Baldwin, 1998).

Likewise, script participant clusters can be induced automatically. Regneri et al. (2011) look at unsupervised induction of script participant clusters from crowdsourced ESDs. They use an integer linear program that makes use of semantic similarity and structural information about which participants typically occur with which events to induce sets of noun phrases that denote the same participant type.

## 2.2 Scripts as Narrative Chains

**Representation.** Another type of representation of script knowledge is based on narrative chains. A narrative chain is a chain of events extracted from a text, in which the protagonist of the text plays a role. In contrast to crowdsourced script knowledge, narrative chains are extracted from large text collections such as news text or blog texts. The left hand side of Figure 2.2 illustrates the induction of two narrative chains from two texts. As preparation,

a dependency parser is run on the text. The protagonist for each text is identified based on a coreference analysis (cf. next section): *Peter* for the upper text and *Mary* for the lower text. A narrative chain is defined as the chain of verbs that have a direct dependency relation to a mentioning of the protagonist. The chain consists of single *events*, each of which is a tuple of the respective verb and the dependency label to the protagonist mention. The right side of Figure 2.2 shows the two chains that are extracted from the texts.

**Models and Evaluation.** The work by Chambers and Jurafsky (2008) established the idea of narrative chains. They extract all coreference chains from Gigaword (Graff, 2002) and define the protagonist of the text to be the discourse entity with the longest coreference chain. Verbs are extracted based on a dependency parse and for each text in Gigaword, one chain is extracted.

To evaluate their script model, they propose the *narrative cloze* task. One event in a narrative chain, i.e. a *(verb, dependency label)* pair, is hidden and needs to be predicted from all possible events. To evaluate their model, they rank all possible events with respect to PMI. To get co-occurrence numbers for computing the PMI, they define that two events are co-occurring if they appear within the same chain in the corpus. As a performance measure, they average over the rank of the correct event for all test instances.

Chambers and Jurafsky (2009) generalize narrative chains by looking not only at the protagonist, but at all participants. A participant corresponds to a coreference chain and is represented with the most common head noun of the chain. They extend their *(verb, dependency label)* representation to encode information about such participants: Each event is represented by a verb and its argument slots (subject, object, and prepositional object), which are filled with participants.

Jans et al. (2012) use the simple narrative chain representation employed by Chambers and Jurafsky (2008), but extract narrative chains irrespective of a protagonist: They also extract chains for discourse referents that do not belong to the longest coreference chain. Also, they look at event bigrams and compute bigram probabilities of events, which they find to perform better than PMI-based measures. They evaluate their model using the average rank as Chambers and Jurafsky (2008), but also report an evaluation based on recall@50.

Pichotta and Mooney (2014) follow Jans et al. (2012) in using bigram probabilities, but they

change the event representation such that there is only one chain per text, similar to Chambers and Jurafsky (2009). An event is represented as a tuple of verb, subject, object and prepositional object. This richer representation contains more information but is also prone to sparsity problems.

Rudinger et al. (2015b) look at the representation based on *(verb, dependency)* tuples, but extend the idea of using probability estimates instead of PMI. They utilize a log-bilinear language model on sequences of events, which leads to large improvements as compared to previous work.

## 2.3 Neural Script Representations

In recent years, neural script knowledge representations have gained considerable popularity. What these models have in common is that they operate on a structure that is similar to narrative chains: To our knowledge, all neural models to date use verbs and their arguments, either in the form of narrative chains or directly extracted from a text. The evaluation of neural script models depends on the exact model formulation and the task. Neural models are trained both on crowdsourced texts and news texts.

Modi and Titov (2014) implement an event embedding model that learns embedding vectors as event representations, based on verbs and their arguments. They use a linear ranker, which learns typical event order from the textual order in both news texts and crowdsourced ESD collections. They evaluate their model on the event ordering task and paraphrasing task proposed by Regneri et al. (2010).

Granroth-Wilding and Clark (2016) use a similar event embedding model, which is based on narrative chains and evaluated on the narrative cloze test. Different to previous work, they introduce a multiple-choice version of the narrative cloze test, in which the correct event needs to be picked from a number of choices.

Pichotta and Mooney (2016) employ LSTM modules to encode events in a narrative chain and to predict the missing element in a narrative cloze test. By using a recurrent architecture, their model is able to utilize longer context for the event prediction.

Weber et al. (2018) implement a hierarchical variational autoencoder and evaluate it on an inverse multiple choice narrative cloze task, in which the complete chain needs to be

identified from several choices, based on the first event. They show that the chains generated by their model are more consistent as compared to previous efforts.

## 2.4 Differences and Commonalities

Narrative chains and paraphrase sets both aim to represent script knowledge but differ in important aspects. These differences imply that narrative chains and paraphrase sets cannot be used interchangeably and that they are for example not equally well suited for applications in downstream tasks.

First, the most striking difference is the way in which script events and participants are represented. While narrative chains represent events based on their surface form - essentially with verb lemmas and dependency relations - paraphrase sets provide realization variants for an event and thus a much richer representation. Also, most narrative chain representations do not provide an explicit representation of script participants. While narrative schemas represent participants, they restrict the representation to a single text, which provides less information than a diverse paraphrase set. This difference in representing events and participants means that narrative chains provide less information for downstream applications, since they essentially only encode ordering information about verbs.

Second, paraphrase sets can only be learned from (monolingual) parallel standard texts, while narrative chains can be extracted from text of any kind. Paraphrase sets are usually induced from a range of texts that talk about a similar scenario and have a similar discourse structure, such as crowdsourced event sequence descriptions (Regneri et al., 2010; Wanzare et al., 2016, 2017) or movie scripts (Regneri and Wang, 2012). This implies that paraphrase sets are usually created on a smaller scale and only on a restricted number of scenarios, while narrative chains do not have this restriction and can be extracted from any large text collection. As a coarse estimation of the actual topical coverage of existing ESD collections, Wanzare et al. (2017) found that three of the most common collections cover approx. 26% of the stories in the ROC data set (Mostafaza deh et al., 2016), a corpus of general everyday-scenario narrations. This means that there is a large space of topics that is not covered by paraphrase sets, and that would thus require further data collection efforts.

Third, however, the fact that narrative chains are often learned from news texts results in

certain restrictions: Script knowledge is usually less apparent and only marginally prominent in the news genre. Scripts address and describe everyday activities, while news texts rarely cover such topics, but rather talk about politics, economics etc. This implies that the genre that narrative chains have been learned on in previous work not always provides a good basis for learning script knowledge: With regard to their discourse mode, news texts are usually classified as *reports* and not as *narrations* (Palmer and Friedrich, 2014). Also, Chambers (2017) note that many verbs in news texts actually don't describe script events at all.

Fourth, certain events are unlikely to be mentioned in texts, even if they naturally occur. Gordon and Van Durme (2013) refer to this effect as *reporting bias* and show that certain events are under-reported in natural texts. This can be attributed to the fact that according to the Gricean maxim of quantity (Grice, 1975), communication should not provide more information than necessary - obvious events are often left out.

Paraphrase sets avoid this restriction to a certain extent, since they are usually learned on very explicit texts. By telling crowdsourcing workers to list all steps that are required for an activity, the resulting texts are as explicit as possible. However, some cases still remain elusive even in such explicit, crowdsourced texts: In *DeScript* for example, the event of putting on the seat belt during a flight is mentioned more than 20 times, while the unbuckling is mentioned only 7 times, although both events should take place equally often.

Lastly, it is hard to compare neural script representations to paraphrase sets or narrative chains. Neural script models are always tailored to the task at hand. To date, to our knowledge, there have been no efforts to evaluate neural script representations in an actual NLU application such as question answering or text summarization.

## 2.5 Other Commonsense Knowledge Representations

In this thesis, the focus is on script knowledge, a specific kind of commonsense knowledge. There has been some research on the wider field of commonsense knowledge, in particular in learning, representing and applying different kinds of commonsense knowledge to various NLP problems. The focus of research on this topic is mainly on a more general kind of knowledge about concepts and entities in the world, and their relations to each other,

organized for example in knowledge graphs. In contrast to this, script knowledge is more focused on temporal order information and event structure. Both general commonsense knowledge in the form of knowledge graphs and script knowledge serve a similar purpose: They both provide background information (of a different kind) and make it possible to draw inferences that go beyond what is mentioned in a text.

Although this thesis focuses only on script knowledge, we provide an overview of recent work on commonsense knowledge as organized in the form of large knowledge graphs in this section because of its similarity to script knowledge. The presented datasets encode different kinds of background knowledge, covering facts about our world (e.g. information about presidents of the US), as well as more basic types of knowledge (water is wet) - or mixtures thereof.

*Cyc* (Lenat, 1995) is a large, manually created commonsense knowledge database. It contains over 300,000 entities, referred to as *individuals*, and *collections*, which are groups of individuals. Cyc comprises of over 3 million relations, which correspond to predicates in first-order predicate logic. Relations encode *functions* (e.g. *capital-of*), which map individuals to other individuals, such as *capital-of(Paris, France)*[1]. Also, *truth functions* are encoded, which map individuals to truth values[2], e.g. *is-a(BillClinton, UnitedStatesPresident)*. Cyc encodes both factual knowledge about our world (e.g. information about US presidents or information about the capital of France) as well as more general assertions ("all trees are plants"). Together with the knowledge base as such, parsing and querying tools are provided for a fast access to the data base. Using the predicate-logic like structure of Cyc, such tools are able to perform simple logical deduction on entities and relations. This predicate-logic based structure of Cyc has however been the center of criticism, since it makes the extension of Cyc complex and requires human workers for this task (Domingos, 2015). Also, the fact that Cyc is not freely available hinders its usefulness for academic research.

*ConceptNet* (Speer et al., 2017) is a large semantic graph that represents general commonsense knowledge about the world. It encodes information about entities and their connec-

[1]Entities and relations actually follow certain syntactic patterns which are omitted here for reasons of brevity.

[2]Numbers are taken from http://www.dave-reed.com/csc550.S03/Presentations/CYC.ppt. Exact and up-to-date numbers are hard to find since the data base is copyrighted and not freely accessible.

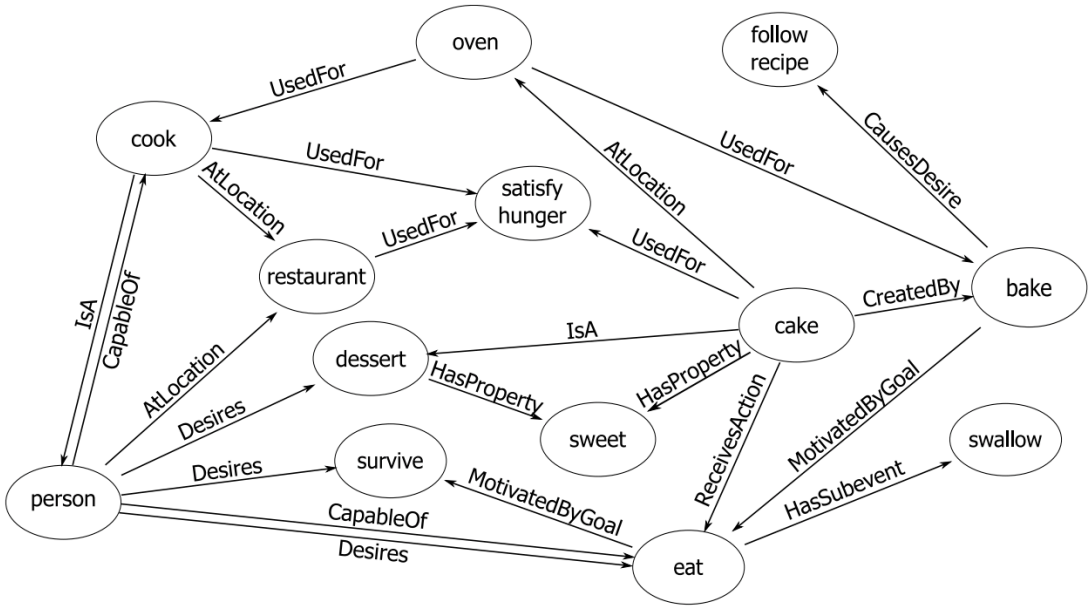



**Figure 2.3:** A small excerpt of *ConceptNet*, taken from Speer and Havasi (2012).

tions to each other (cf. Figure 2.3), similar to *WordNet* (Fellbaum, 1998). It contains 21 million edges, encoding the entities, and over 8 million nodes, encoding relations between the entities. ConceptNet was mainly created automatically by extracting and parsing information from various sources, such as the *OMCS* project (Singh et al., 2002), *Wiktionary*[3], *Open Multilingual WordNet* (Bond and Foster, 2013), *OpenCyc* (Lenat, 1995) (a small subset of the full database, cf. next paragraph) and *DBPedia* (Auer et al., 2007). As a consequence, it encodes very different types of knowledge. Since ConceptNet is mostly automatically created, it is easily extendible, in contrast to Cyc. It does not allow precise predicate-logic based reasoning, but is better suited for statistical modeling and has thus been more popular recently. Also, its free availability is a clear advantage compared to Cyc.

*WebChild* (Tandon et al., 2017) is a knowledge base that was automatically created from web contents. It contains over 2 million adjectives and nouns that are mapped to WordNet senses for disambiguation, and that are connected with over 18 million assertions based on predicates such as *hasShape* or *hasTaste*. In contrast to other commonsense knowledge databases, WebChild is tightly connected to WordNet, providing word sense disambiguation for its terms. WebChild mostly encodes basic commonsense knowledge, such as the

[3] https://www.wiktionary.org/

fact that *a die is a type of cube*.

*NELL* (Carlson et al., 2010; Mitchell et al., 2015, 2018) is similar to WebChild in the sense that it contains commonsense assertions collected from the web. Unlike WebChild, NELL is based on a machine learning system that constantly scans the web and refines and extends its knowledge base based on a manually created seed set.

*DBPedia* (Lehmann et al., 2015) is a knowledge graph that was automatically extracted from *Wikipedia* and contains 400 million facts describing 3.7 million entities. In contrast to other data bases, DBPedia is multilingual, including information from 111 different language editions of Wikipedia and only focuses on factual knowledge about named entities that are mentioned in Wikipedia.

# Part II

# Identifying Scripts in Natural Texts

Chapter 3

# Script Parsing - Background

The focus of the second part of this thesis is on the task of analyzing text-level event structure. We address *script parsing*, the automatic mapping of narrative texts to scripts. Script parsing is the most central prerequisite for leveraging script knowledge for end-to-end tasks and applications of natural language understanding, since it links the script representation to the text and makes paraphrase and ordering information of the script representation accessible. In Section 3.1, we explain the basic idea of script parsing and motivate the task by showing application perspectives. In Section 3.2, we move our focus to a more naturalistic example and illustrate the difficulty of the task with a blog story text. Since the task bears major challenges, we simplify it for this dissertation. These simplifications and our resulting contributions are described in Section 3.3. Lastly, we describe related work in Section 3.4.

## 3.1 Basic Idea and Motivation

Figure 3.1 illustrates the basic idea of script parsing. A fragment of a story about BAKING A CAKE is shown in the upper part of the figure; with a script fragment in the lower part. In this dissertation, our script parser will utilize a script representation in the form of paraphrase sets (cf. Section 2.1), in which events and participants are represented as sets of short formulation variants. The idea of script parsing is to identify verbs in a text that instantiate events

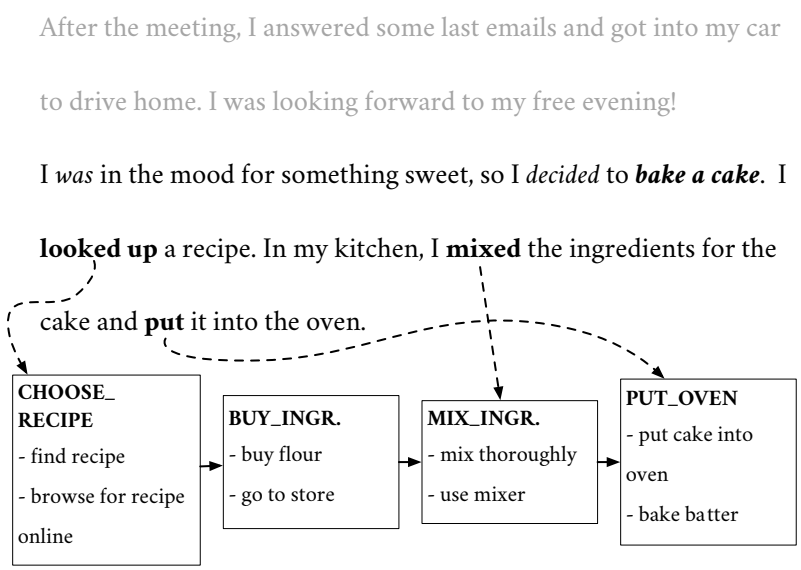



**Figure 3.1:** An example of script parsing with a story fragment and an excerpt of the BAKING A CAKE script.

of a given script and align them with the respective event paraphrase sets. Script parsing subsumes the task of identifying and labeling script participants in the text, which for better readability is not shown in the figure.

In the example text, *look up*, *mix* and *put* mention script events that appear in the BAKING A CAKE script. Therefore, they are aligned with the respective paraphrase sets (CHOOSE_RECIPE, MIX_INGREDIENTS and PUT_OVEN). The auxiliary verb *was* does not describe an event, so a script parsing model will not align it. *Decide* describes a specific action that can generally appear in many scenarios. In contrast to the other events, it is independent of a concrete script and is thus not aligned. *Bake a cake* plays a special role: Rather than describing a concrete event, it evokes and activates the BAKING A CAKE script. A script parser can use such script-evoking elements to identify the script that is addressed in the text.

As mentioned, a full script parser also identifies participant mentions in a text and labels them with participant type labels. It should find that *flour*, *chocolate* and *ingredients* refer to the INGREDIENTS participant type, while *chocolate cake* for example refers to the CAKE participant.

Script parsing consists of five sub-tasks. These tasks are depicted in Figures 3.2 and 3.3, on the same text as in Figure 3.1.

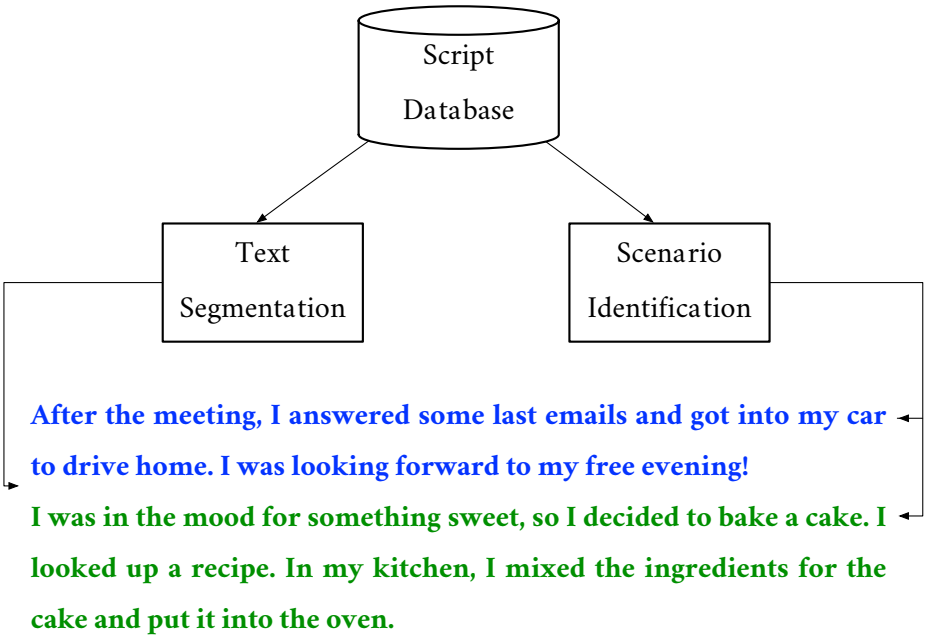



**Figure 3.2:** Text Segmentation and Scenario identification based on a script database.

- **Text Segmentation.** One important step of script parsing is to identify parts of a text that talk about a specific scenario. In Figure 3.2, this step corresponds to identifying that the blue and green parts address different scripts.

- **Scenario Identification.** Identifying the relevant script scenario is closely connected to the text segmentation step. A script parser should make use of script-evoking elements such as *bake a cake* in the example to activate the BAKING A CAKE script and make it accessible for script parsing. In Figure 3.2, this step corresponds to assigning different scenario labels to the blue and green parts (WORKING IN A JOB and BAKING A CAKE, respectively).

- **Event Verb Identification.** Given a text fragment that instantiates a specific script, the next step is to identify verbs that denote script events. This corresponds to finding that *was* and *decided* should not be aligned, and that *looked up*, *mixed* and *put* denote events, in Figure 3.3.

- **Event Type Identification.** The most central step is the identification of the event type of verbs that were previously identified as denoting a script event, i.e. the actual alignment of verbs in the text with event paraphrase sets. This corresponds to assigning

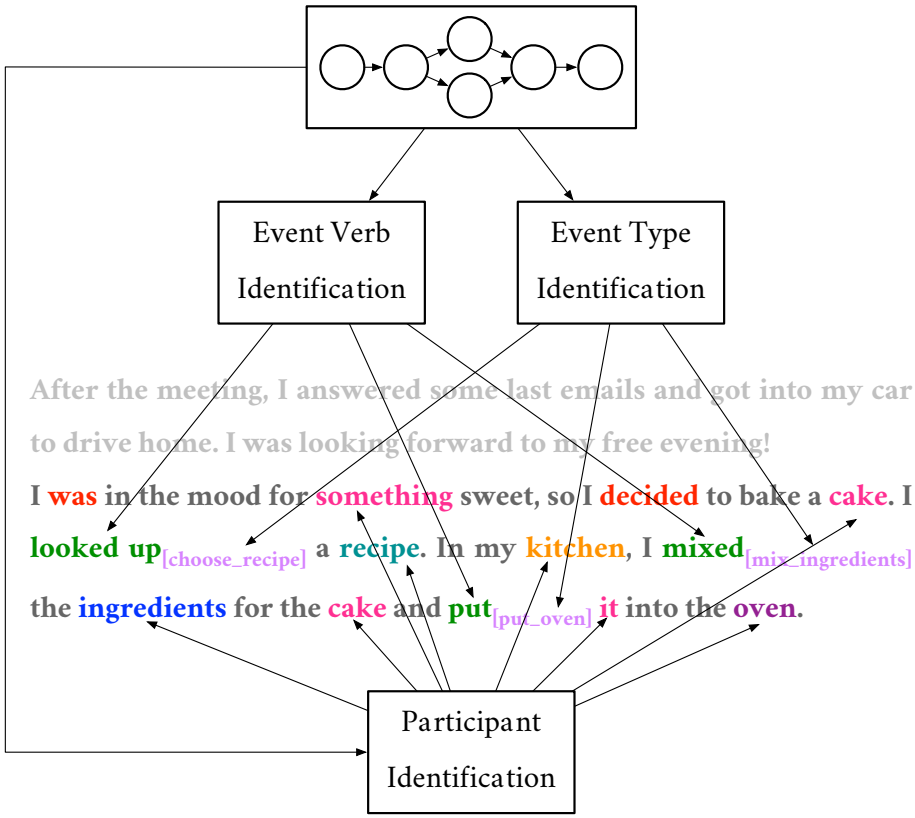



**Figure 3.3:** Event Verb Identification, Event Type Identification and Participant Identification.

the purple labels in Figure 3.3.

- **Participant Identification.** Script participant mentions and their participant type should be identified. The participant type labels are not shown in the figure for better readability.

In human language understanding, these five subtasks are integrated rather than performed as single steps. The subtasks are thus not as strictly separable as indicated here. Information from one task might be useful for other tasks, so ideally, a script parser is an integrated system in which information can flow from one component to the other. An example is that participant information is useful for scenario identification, since there are cases in which the scenario is evoked only based on a participant or event mention (s. Section 3.2). Knowing about a participant type can thus be helpful for finding the current scenario.

Script parsing only makes sense as a task if the underlying script representation is based on event types: The alignment of narrative chain events with a text is trivial, since events are

represented with verb lemmas rather than complex types in a narrative chain (cf. Chapter 2). Early script parsers (Cullingford, 1978) used a hand-coded event type representation, which enabled script parsing. However, a small, hand-crafted script database does not allow script parsing on a larger scale. In contrast, scalable script parsing is facilitated by a representation based on paraphrase sets. Large paraphrase sets with a high coverage and a high linguistic diversity make the parsing task easier, since they provide richer and more training material for a parser (cf. Chapter 5). Also, when the script is aligned with the text, the paraphrase information on event and participant level is accessible for concrete text processing applications. Examples include question answering, textual entailment, coreference resolution, text generation and others, as discussed in Chapter 1.

## 3.2 Script Parsing of Naturalistic Texts

The text in Figure 3.1 illustrated the basic idea and intuition behind script parsing. In this section, we look at challenges for script parsing that arise on more naturalistic texts.

The text in Figure 3.4 is a story about everyday activities, taken from the Spinn3r corpus (Burton et al., 2009), a large collection of blog stories crawled from various web pages. This text illustrates some of the difficulties for script parsing on naturalistic texts, but does not even show the most complex cases. As discussed in Chapter 2, there are some genres, such as newswire texts, which refer to scripts only briefly and infrequently, so parsing such a text will be more challenging.

The text in Figure 3.4 illustrates some of the challenges for text segmentation and scenario identification:

- **Multiple Scenarios.** There can be a large number of scenarios addressed in a single text. Within one text, the scenario can change several times. In Figure 3.4, at least three scenarios are instantiated, as indicated by the different colours (VISITING A CONVENTION, HAVING LUNCH and TAKING A TOUR). There could also be cases in which a short text chunk is an excursion into a different topic, such that the chunk that instantiates a scenario is discontinuous.

- **Scenario Granularity.** It is not clearly defined which events and participants belong

Yesterday morning, we couldn't wait to go over to the Salt Palace convention center to pick up our new convention bags! Inside were plenty of goodies, including the brand-new catalog and three new stamp sets! Then I spent almost three hours in line to get into Memento Mall. This is where Stampin' Up! sells souvenier type stuff. I bought a super cute new shirt with trimming from the new Windsor Knot designer series papers, a CD-rom with samples from convention classes, charms, cute chapsticks in our new Designer Series prints, and other fun things. Next, I *went to lunch* with Robin and Dallas. I really liked the Coca-Cola pork loin sandwich I *had for dinner* Tuesday night, so I recommended we go back to the Nauvoo Cafe for lunch. I'm talking real sandwiches. With real homemade bread and they carve the meat for your sandwich right in front of you. Yum! After lunch, the others *went on the tour* of the Stampin' Up' canvas in Riverton. I enjoyed wandering through Temple Square and got some pics of the temple, the other beautiful buildings, and the surrounding gardens. Then, I saw the LDS conference center through the trees and decided *to go take a tour*. The architecture and artwork are amazing!

**Figure 3.4:** Example text from the Spinn3r blog stories corpus (Burton et al., 2009). Script events are underlined, script-evoking events are marked in *italics*.

to a particular scenario and which do not. The granularity of scenarios depends on the script data base. In the green text, the HAVING LUNCH scenario is addressed, so a parser needs to know that a different script has to be activated. Alternatively, the HAVING LUNCH script could be seen as a part of VISITING A CONVENTION, so the parser has the options to either activate this script or not.

- **Script Activation.** There are often no expressions that clearly evoke a script. In the beginning of the text in Figure 3.4, there is no expression that evokes the VISITING A CONVENTION scenario, but the first action that is mentioned is an event from the scenario: GOING TO THE CONVENTION CENTER. The parser will use the participant CONVENTION CENTER (and some surrounding material) to activate the correct script.

- **Distractors.** There can be distracting expressions, in which a script is clearly addressed and activated, but not referred to further. In the green segment in Figure 3.4, the expression *had for dinner* evokes the HAVING DINNER script, but there are no

references to events or participants of the script in the text. In this case, the parser should not activate a different script representation. *Taking a tour* is mentioned twice in the orange part, so the TAKING A TOUR script is activated. However, no relevant events from the script are mentioned.

For the identification of event verbs and event types, the following difficulties arise.

- **Sparse Event Mentions**: From the text, it becomes apparent that there are usually only few relevant events addressed, because most of what happens in a script is assumed to be common ground and implicitly understood by the reader. This makes it hard to use rich script information or material from the text, because the parts of a text that actually address a script can be short and sparse. The green text for example describes a full instance of the HAVING LUNCH script, but mentions only two events, instantiated by *recommend* and *carve*, respectively.

- **The Role of Ordering Information.** We generally assume that ordering information is central to script parsing: The typical order of events in a script scenario should guide the alignment process and inform the identification of event types. However, the natural order of many events is very flexible, as can be seen in the blue text in Figure 3.4: *spending time in a line*, *buying a shirt* and *get into a hall* can all happen in arbitrary order. It is thus difficult to utilize temporal information in this case.

- **Event Identification.** There are many verbs that should not be aligned with the script at all, for example *could*, *wait* or *were*. While auxiliaries usually don't describe script events, it is far from trivial to find out that *wait* in this case is not relevant for the current script. WAIT is a plausible event when VISITING A CONVENTION, but in this case, the verb does not describe the respective event but is used as a light verb.

## 3.3 Script Parsing in the Scope of this Thesis

In this thesis, we provide the first script parsing model trained on paraphrase sets. As shown in Section 3.2, aligning script structures with texts is a complex task. Providing a fully supervised, generally applicable script parser that tackles all of the aforementioned tasks as shown

in Section 3.1 would go beyond the scope of this dissertation. Thus, we simplify the task and concentrate on some parts of the full task only. We find that even a simplified script parsing task is a complex challenge and identify relevant problems.

- As a first simplification, we work with a data set of narrative texts that instantiate a single script only, i.e. that contain a story situated in a single scenario, described in Chapter 4. This prevents the need to perform a text segmentation step. We concentrate on only 10 scenarios, so an automated scenario identification is not necessary. However, there has been work on segmenting blog texts and identifying script scenarios in text segments (Wanzare et al., 2019), which was conducted in the project research group and which is described as related work in Section 3.4.

- In this dissertation, we will only cover event-based script parsing, and will not look at participant identification. It is important to note that participant information is central for the event alignment: Our script parser, presented in Chapter 6, makes use of participant information in the form of noun heads, but we do not perform an explicit participant labeling step. There has been work on this task (Kampmann et al., 2015), which is also described in Section 3.4.

- We assume that event verb identification and the event type identification are separate steps and that the two steps do not inform each other. In our script parsing model, we treat both tasks independently and implement two independent models - one for each task - which are then combined for an end-to-end evaluation, in Chapter 6.

As depicted in Figure 3.3, we assume that a script parsing model can be trained only on paraphrase sets and does not need an alignment gold standard for training. As we will see in Chapter 4, the annotation of such a gold standard is time consuming, difficult and expensive: It requires skilled and trained expert annotators and scenario-specific guidelines. Script data in the form of paraphrase sets in contrast are relatively cheap to acquire on a larger scale via crowdsourcing, and paraphrase clusters can be induced semi-automatically (Regneri et al., 2010; Wanzare et al., 2017).

## 3.4 Related Work

Event structure is a prominent topic in NLP. Semantic role labelers (Gildea and Jurafsky, 2002; Palmer et al., 2010) are well-established tools for the analysis of the internal structure of event descriptions: They are used to assign semantic frames to verbs and other phrases, which makes frame semantic information available for natural language understanding applications.

In recent years, modeling relations between events has gained increasing attention. An example is research on event coreference (Bejan and Harabagiu, 2010; Lee et al., 2012). The task is related to "classical" coreference resolution based on NPs, but with the aim to identify mentions of *events* that refer to the same action in the real world.

Another example is temporal event ordering in texts (Ling and Weld, 2010), as well as cross-document event ordering (Minard et al., 2015, inter alia), where temporal relations between events are modeled. Systems take into account discourse structure in order to learn the actual temporal order of events from the order of their mentioning in a newswire text.

Since this thesis provides the first actual script parsing model that is trained on scripts as represented by crowdsourced paraphrase sets, there is no directly related work that we can compare our work to.

Early script models such as SAM (Cullingford, 1978) included a script parsing module. SAM was a computer program that was similar to the architecture proposed in Figures 3.2 and 3.3. It had script activation modules that could identify a script in a text based on an evoking expression or a script event. It would incrementally process a text and assign entities to predefined participant slots. Also, it could detect mentions of script events and map them to an internal script representation. As mentioned earlier, SAM and its successors were based on hand-written rules and worked only on a number of artificial texts. Therefore, a direct comparison to our model, which is trained in a supervised fashion and which can in principle be extended to any script domain, if crowdsourced paraphrase sets are available, is not possible.

Kampmann et al. (2015) present a script parsing model for participants. They implement several models for the automatic labeling of coreference chains and noun heads with participant types. Their models use a script representation based on paraphrase sets and utilize

different textual similarity measures to find matching participant type labels.

Wanzare et al. (2019) present a system for text segmentation and scenario identification. They use topic models to segment blog stories from the Spinn3r collection (Burton et al., 2009) and use a multilayer perceptron that is trained on texts from MCScript (s. Chapter 8) to assign scenario labels to the segments. These tasks corresponds to the text segmentation and scenario identification steps in Figure 3.2. Instead of using ESDs for the scenario identification, they train their system on full narrative texts on predefined script scenarios.

As pointed out in the introduction, script knowledge is in some respects similar to frame semantics. In Chapter 6, we show for example that a frame analysis is beneficial for script parsing. The idea of semantic role labeling is to assign predefined frame labels to verbs, with the main difference being that script events are defined specific to a scenario, while frames are defined irrespective of a scenario. Script parsing can inherently be understood as a discourse-level task, because temporal ordering information about events beyond sentences is an important information source. Recent work found that discourse information, and thus, also script knowledge, is beneficial for semantic role labeling. This is exemplified e.g. by Roth and Lapata (2015), who compute features based on the discourse context and show that these improve a semantic role labeling model.

Chapter 4

# Script Parsing Data - The InScript Corpus

In this chapter, we describe a data collection that was conducted in order to evaluate our script parser (s. Chapter 6). Figure 4.1 again illustrates the core idea of script parsing: Verbs in a text that instantiate events of a given script are aligned with the script events, in order to access information that is associated with the script.

In Chapter 3, we illustrated the complexity of script parsing. Since in this dissertation project, we provide the first scalable script parsing model, we simplify the task to make it more tractable. In particular, we present *InScript*, a corpus of crowdsourced naturalistic texts which only talk about a single scenario. This eliminates the need for text segmentation. Additionally, we only concentrate on a small number of predefined scenarios, so a scenario identification step is not necessary. InScript is used as the evaluation data set for our script parsing model in the following chapters. Texts in InScript have certain properties that make them interesting for the evaluation of script models:

**Domain.** Texts in InScript are narrations about everyday scenarios. Only one scenario is addressed per text, so an evaluation of our script parsing model based on one single scenario is possible.

**Explicitness.** Many events that take place during the everyday situation are mentioned, as well as participants that play a role in the script. In other words, texts in InScript are explicit

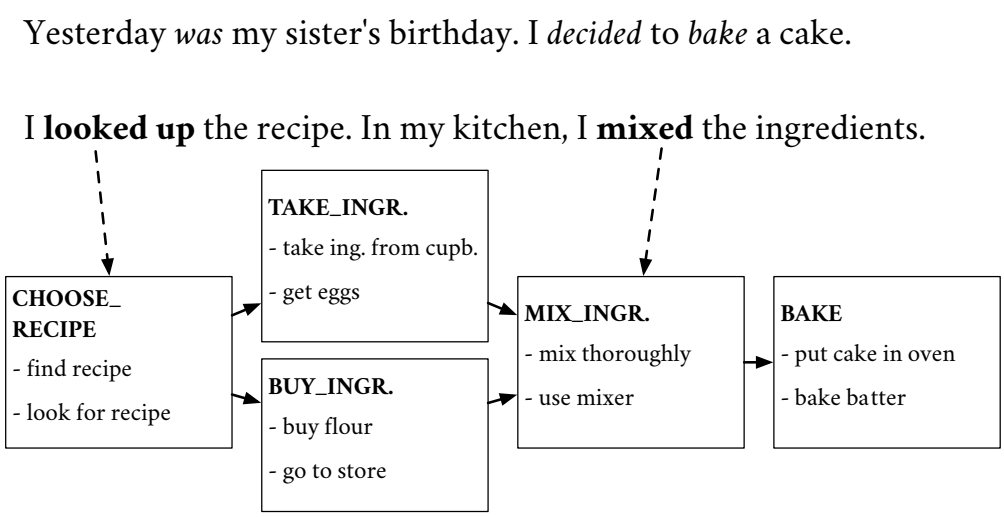



**Figure 4.1:** An example of script parsing with an excerpt of the BAKING A CAKE script and a story snippet.

with regard to events and participants that appear in the script in question.

**Linguistic Complexity.** Texts in InScript are linguistically simple and yet naturalistic. Our script parser therefore does not need to deal with syntactically and semantically complex problems, but can focus on identifying script events.

Importantly, the scenarios of the texts in InScript also appear in DeScript, such that a mapping of a script representation learned from DeScript to texts from InScript is possible. Together with the crowdsourced texts, we provide an annotation of verbs and nouns with event and participant type labels, respectively. The event type labels in DeScript are harmonized with the ones used in the annotation on InScript: Script event type labels that are used in the annotation on InScript correspond to clusters that are found in DeScript. This makes it possible to associate each annotated event-denoting verb in InScript with a paraphrase cluster in DeScript, as depicted in Figure 4.1.

InScript was collected as part of a larger research effort and also fulfilled other purposes in the scope of the project. Modi et al. (2017) used InScript as an evaluation dataset for a referent cloze task, i.e. the task of predicting an upcoming discourse referent. For this task, they required hand-annotated coreference chains on all noun heads of the corpus, which is why we also conducted a full coreference annotation on all texts. In the scope of this thesis, we

use these coreference annotations to resolve pronouns for our script parsing experiments in Chapter 6.

The structure of this chapter is as follows:

- We present InScript, a corpus of 910 crowdsourced narrative stories about 10 different everyday scenarios. In the following section, we first describe the data collection process of InScript via crowdsourcing and provide data statistics.
- Together with the texts, we provide a full annotation of all verbs with script event type labels and of all nouns with script participant type labels. We also labeled coreference chains on all noun heads. In Section 4.2, we describe the annotation.
- In the last section, we provide a thorough analysis of the annotation. We perform a linguistic comparison of our data with DeScript in terms of the vocabulary and the complexity of event descriptions.

## 4.1 Data Collection

### 4.1.1 Collection via Amazon Mechanical Turk

We selected 10 scenarios from different available scenario lists (e.g. Regneri et al. (2010), Raisig et al. (2009), and the OMICS corpus Singh et al. (2002)), including scripts of different complexity (Taking a bath vs. Flying in an airplane) and specificity (Riding a public bus vs. Repairing a flat bicycle tire). For the full scenario list see Table 4.1. The selected scenarios are part of the scenarios of the script data given in DeScript, so an evaluation of a script parsing system on the texts is possible.

Texts were collected via crowdsourcing on Amazon Mechanical Turk (MTurk[1]), which provides an opportunity to present an online task to humans. In order to gauge the effect of different MTurk instructions on our task, we first conducted pilot experiments with different variants of instructions explaining the task. We finalized the instructions for the full data collection, asking the turkers to describe a scenario in form of a story *as if explaining it*

[1] www.mturk.com

*to a child* and to use a minimum of 150 words. The selected instruction variant resulted in comparably simple and explicit scenario-related stories.

In total, 190 turkers participated. All turkers were living in the USA and native speakers of English. We paid $0.50 per story to each turker. On average, the turkers took 9.37 minutes per story with a maximum duration of 17.38 minutes.

### 4.1.2 Data Statistics

Statistics for the corpus are given in Table 4.1. On average, each story has a length of 12 sentences and 217 words with 98 word types on average. Stories are coherent and concentrate mainly on the corresponding scenario. Neglecting auxiliaries, modals and copulas, on average each story has 32 verbs, out of which 58% denote events related to the respective scenario. As can be seen in Table 4.1, there is some variation in stories across scenarios: The FLYING IN AN AIRPLANE scenario, for example, is the most complex in terms of the number of sentences, tokens and word types that are used.

This is probably due to the inherent complexity of the scenario: Taking a flight, for example, is more complicated and takes more steps than taking a bath. The average count of sentences, tokens and types is also very high for the BAKING A CAKE scenario. Stories from the scenario often resemble cake recipes, which usually contain very detailed steps, therefore people tend to give more detailed descriptions.

For both FLYING IN AN AIRPLANE and BAKING A CAKE, the standard deviation is higher in comparison to other scenarios. This indicates that different turkers described the scenario with a varying degree of detail and can also be seen as an indicator for the complexity of both scenarios.

In comparison, texts from the TAKING A BATH and PLANTING A TREE scenarios contain a smaller number of sentences and fewer word types and tokens. Both planting a tree and taking a bath are simpler activities, which results in generally less complex texts.

The average pairwise word type overlap can be seen as a measure of lexical variety among stories: If it is high, the stories resemble each other more. We can see that stories in the FLYING IN AN AIRPLANE and BAKING A CAKE scenarios have the highest values here, indicating

| Scenario Name | #Stories | Avg. Sentences Per Story | Avg. Word Types Per Story | Avg. Word Count Per Story | Avg. Word Type Overlap |
|---|---|---|---|---|---|
| Riding in a public bus (Bus) | 92 | 12.3 (4.1) | 97.4 (23.3) | 215.1 (69.7) | 35.7 (7.5) |
| Baking a cake (Cake) | 97 | 13.6 (4.7) | 102.7 (23.7) | 235.5 (78.5) | 39.5 (8.1) |
| Taking a bath (Bath) | 94 | 11.5 (2.6) | 91.9 (13.1) | ***197.5 (34.5)*** | 37.9 (6.3) |
| Going grocery shopping (Grocery) | 95 | 13.1 (3.7) | 102.9 (19.9) | 228.3 (58.8) | 38.6 (7.8) |
| Flying in an airplane (Flight) | 86 | **14.1 (5.6)** | **113.6 (30.9)** | **251.2 (99.1)** | **40.9 (10.3)** |
| Getting a haircut (Haircut) | 88 | 13.3 (4.0) | 100.6 (19.3) | 227.2 (63.4) | 39.0 (7.9) |
| Borrowing a book from the library (Library) | 93 | 11.2 (2.5) | ***88.0 (14.1)*** | 200.7 (43.5) | 34.9 (5.5) |
| Going on a train (Train) | 87 | 12.3 (3.4) | 96.3 (19.2) | 210.3 (57.0) | 35.3 (6.9) |
| Repairing a flat bicycle tire (Bicycle) | 87 | 11.4 (3.6) | 88.9 (15.0) | 203.0 (53.3) | 33.8 (5.2) |
| Planting a tree (Tree) | 91 | ***11.0 (3.6)*** | 93.3 (19.2) | 201.5 (60.3) | ***34.0 (6.6)*** |
| *Average* | *91* | *12.4* | *97.6* | *216.9* | *37.0* |

**Table 4.1:** Corpus statistics for different scenarios (standard deviation given in parentheses). The maximum per column is highlighted in **boldface**, the minimum in ***boldface italics***.

that most turkers used a similar vocabulary in their stories.

In general, the response quality was good. Only 9% of the stories were discarded as they lacked the quality we required. This mainly affected stories that were ungrammatical or contained many typos, as well as instructional texts, i.e. texts that were not narrative but gave instructions. In total, we selected 910 stories for annotation.

## 4.2 Annotation

This section deals with the annotation of the data. The annotation was an iterative process of corpus annotation and required a refinement of the schema. This refinement was necessary due to the complexity of the annotation. We will first describe the final annotation schema. Then, we will describe the schema refinement. Both versions of the annotation schema are given in the appendix, in Chapters A and B.

### 4.2.1 Annotation Schema

For each of the scenarios, we designed a specific annotation template. A *script template* consists of scenario-specific event and participant labels. An example of a template is shown in Table 4.2. All NP heads in the corpus were annotated with a participant label; all verbs were annotated with an event label. For both participants and events, we also offered the label UNCLEAR if the annotator could not assign another label. Coreference chains between NPs were additionally annotated. Thus, the process resulted in three layers of annotation: event types, participant types and coreference annotation. These are described in detail below. All annotations were conducted by computational linguistics students. Figure 4.2 shows a screenshot of a complete participant and event type annotation.

#### Event Type

As a first layer, we annotated event types. In Chapter 3, we argued that there are three categories of verbs relevant to script parsing: Events that denote script events, evoking elements, and verbs that do not denote script events. While we provide labels for the first two classes, the class of non-events cannot be annotated as easily, as explained below.

**Verbs that denote script events.** In our annotation guideline, most labels for verbs that denote script events are scenario-specific. We listed script event types in the scenario-specific templates. Such labels include for example SCREV_CLOSE_DRAIN in TAKING A BATH as in Example 1 (see Table 4.2 for a complete list for the TAKING A BATH scenario).

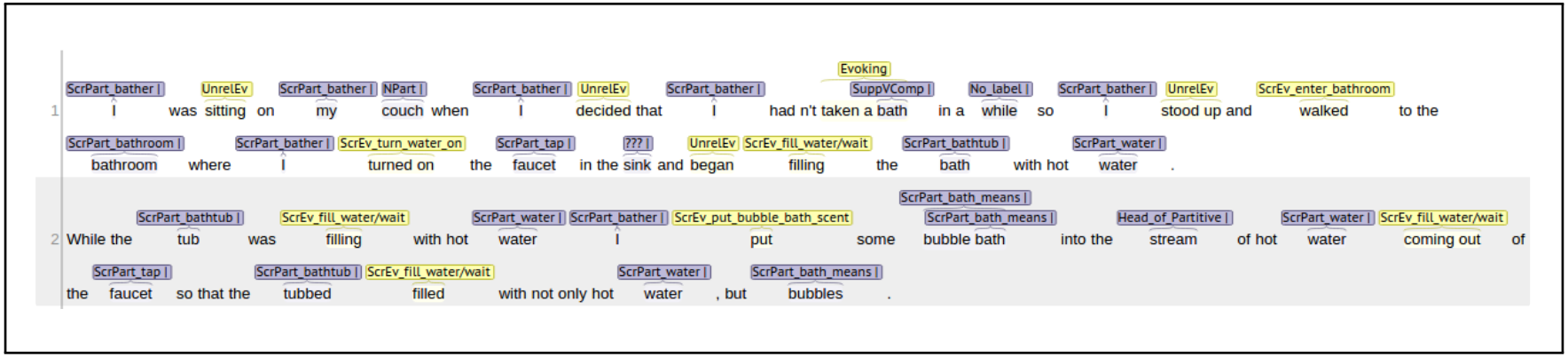



**Figure 4.2:** Sample event and participant annotation for the TAKING A BATH script.

| Event types | Participant types |
|---|---|
| ScrEv_take_clean_clothes | **ScrPart_bath** |
| ScrEv_prepare_bath | ScrPart_bath_means |
| ScrEv_enter_bathroom | ScrPart_bather |
| ScrEv_turn_water_on | ScrPart_bathroom |
| ScrEv_check_temp (temperature) | ScrPart_bathtub |
| ScrEv_close_drain | ScrPart_body_part |
| ScrEv_wait | ScrPart_clothes |
| ScrEv_turn_water_off | ScrPart_drain |
| ScrEv_put_bubble_bath_scent | ScrPart_hair |
| ScrEv_undress | ScrPart_hamper |
| ScrEv_sink_water | ScrPart_in-bath_entertainment (candles, music, books) |
| ScrEv_relax | **ScrPart_plug** |
| ScrEv_apply_soap | **ScrPart_shower** (as bath equipment) |
| ScrEv_wash | ScrPart_tap (knob) |
| ScrEv_open_drain | ScrPart_temperature |
| ScrEv_get_out_bath | ScrPart_towel |
| **ScrEv_get_towel** | ScrPart_washing_tools (washcloth, soap) |
| ScrEv_dry | ScrPart_water |
| ScrEv_put_after_shower | |
| ScrEv_get_dressed | |
| ScrEv_leave | |
| ScrEv_air_bathroom | |

**Table 4.2:** Bath scenario template (labels added in the second phase of annotation are marked in bold).

(1) I start by *closing*$_{\text{ScrEv\_close\_drain}}$ the drain at the bottom of the tub.

Additionally, we have the label ScrEv_other across all scenarios. This is used for script events that belong to the scenario, but have an event type that occurs very infrequently (for details, see Section 4.2.4). We used the label "other" because the event classification would become too fine-grained otherwise.

*Example:* After I am dried I put my new clothes on and *clean up*$_{\text{ScrEv\_other}}$ the bathroom.

**Verbs that evoke a script.** To mark script-evoking expressions, we use the label EVOKING, as shown in Example 2.

(2) Today I *took a bath*$_{\text{EVOKING}}$ in my new apartment.

**Other verbs.** In InScript, we found that it would not make sense to add a label for verbs that do not denote script events, since there are only few such verbs in the very explicit InScript texts. However, we found a larger number of verbs that describe script events that are only loosely connected to the script scenario in question.

To properly label such events, additional templates on different scenarios would be required. Since it is difficult to estimate which scenarios are relevant here, we simplified this case and just used two additional labels, as listed below:

- RELNSCREV. Related non-script event. An event that *can* plausibly happen during the execution of the script and is related to it, but is not part of the script.
  *Example:* After finding what I wanted to wear, I went into the bathroom and *shut*$_{\text{RELNSCREV}}$ the door.

- UNRELEV. An event that is not related to the current script. This class also subsumes verbs that do not denote events at all.
  *Example:* I sank into the bubbles and *took*$_{\text{UNRELEV}}$ a deep breath.

### Participant Type

As in the case of the event type labels, there are two kinds of participant labels: general labels and scenario-specific labels. The latter are part of the scenario-specific templates, e.g. SCRPART_DRAIN in the TAKING A BATH scenario, as can be seen in Example 3.

(3) I start by closing the *drain*$_{\text{SCRPART\_DRAIN}}$ at the bottom of the tub.

The general labels used across all scenarios mark noun phrases with scenario-independent features. These are the following general labels:

- ScrPart_other. A participant that belongs to the scenario, but its participant type occurs only infrequently.
  *Example:* I find my *bath mat*$_{\text{ScrPart\_other}}$ and lay it on the floor to keep the floor dry.

- NPart. Non-participant. A referential NP that does not belong to the scenario.
  *Example:* I washed myself carefully because I did not want to spill water on the *floor*$_{\text{NPart}}$.

- SuppVComp. A support verb complement. For further discussion of this label, see Section 4.2.5.
  *Example:* I sank into the bubbles and took a deep *breath*$_{\text{SuppVComp}}$.

- Head_of_Partitive. The head of a partitive or a partitive-like construction. For a further discussion of this label see Section 4.2.5.
  *Example:* I grabbed a *bar*$_{\text{Head\_of\_Partitive}}$ of soap and lathered my body.

- No_label. A non-referential noun phrase that cannot be labeled with another label.
  *Example:* I sat for a *moment*$_{\text{No\_label}}$, relaxing, allowing the warm water to sooth my skin.

All NPs labeled with one of the labels SuppVComp, Head_of_Partitive or No_label are considered to be non-referential. No_label is used mainly in four cases in our data: non-referential time expressions (*in a* ***while****, a million* ***times*** *better*), idioms (*no* ***matter*** *what*), the non-referential "it" (***it*** *felt amazing*, ***it*** *is better*) and other abstracta (*a* ***lot*** *better, a little* ***bit***).
In the first annotation phase, annotators were asked to mark verbs and noun phrases that have an event or participant type, that is *not* listed in the template, as MissScrEv/ MissScrPart (missing script event or participant, resp.). These annotations were used as a basis for extending the templates (see Section 4.2.4) and later replaced by newly introduced labels or ScrEv_other and ScrPart_other respectively.

#### Coreference Annotations

All noun phrases were annotated with coreference information indicating which entities denote the same discourse referent. The annotation was done by linking heads of NPs (see

Example 4, where the links are indicated by coindexing). As a rule, we assume that each element of a coreference chain is marked with the same participant type label.

(4) $I_{\text{COREF1}}$ washed $my_{\text{COREF1}}$ entire $body_{\text{COREF2}}$, starting with $my_{\text{COREF1}}$ $face_{\text{COREF3}}$ and ending with the $toes_{\text{COREF4}}$. $I_{\text{COREF1}}$ always wash $my_{\text{COREF1}}$ $toes_{\text{COREF4}}$ very thoroughly ...

The assignment of an entity to a referent is not always trivial, as is shown in Example 5. There are some cases in which two discourse referents are grouped in a plural NP. In the example, *those things* refers to the group made up of *shampoo*, *soap* and *sponge*. In this case, we asked annotators to introduce a new coreference label, the name of which indicates which referents are grouped together (COREF_GROUP_WASHING_TOOLS). All NPs are then connected to the group phrase, resulting in an additional coreference chain.

(5) $I_{\text{COREF1}}$ made sure that $I_{\text{COREF1}}$ have $my_{\text{COREF1}}$ $shampoo_{\text{COREF2 + COREF\_GROUP\_WASHING\_TOOLS}}$, $soap_{\text{COREF3 + COREF\_GROUP\_WASHING\_TOOLS}}$ and $sponge_{\text{COREF4 + COREF\_GROUP\_WASHING\_TOOLS}}$ ready to get in. Once $I_{\text{COREF1}}$ have those $things_{\text{COREF\_GROUP\_WASHING\_TOOLS}}$ $I_{\text{COREF1}}$ sink into the bath. ... $I_{\text{COREF1}}$ applied some $soap_{\text{COREF3}}$ on $my_{\text{COREF1}}$ body and used the $sponge_{\text{COREF4}}$ to scrub a bit. ... $I_{\text{COREF1}}$ rinsed the $shampoo_{\text{COREF2}}$.

Example 5 thus contains the following coreference chains:

(6) COREF1: I → I → my → I → I → I → my → I
COREF2: shampoo → shampoo
COREF3: soap → soap
COREF4: sponge → sponge
COREF_GROUP_WASHING_ TOOLS: shampoo → soap → sponge → things

### 4.2.2 Development of the Schema

The templates were carefully designed in an iterated process. For each scenario, we designed preliminary versions of the template based on an inspection of some stories; for a subset of the scenarios, preliminary templates developed at our department for a psycholinguistic experiment on script knowledge were used as a starting point. Subsequently, we manually

annotated 5 randomly selected texts for each of the scenarios based on the preliminary template. Necessary extensions and changes in the templates were discussed and agreed upon. Most of the cases of disagreement were related to the granularity of the event and participant types. We agreed on the *script-specific functional equivalence* as a guiding principle. For example, reading a book, listening to music and having a conversation are subsumed under the same event label in the FLIGHT scenario, because they have the common function of in-flight entertainment in the scenario. In contrast, we assumed different labels for the cake tin and other utensils (bowls etc.), since they have different functions in the BAKING A CAKE scenario and accordingly occur with different script events.

Note that scripts and templates as such are not meant to describe an activity as exhaustively as possible and to mention all steps that are logically necessary. Instead, scripts describe cognitively prominent events in an activity. An example can be found in the FLIGHT scenario. While more than a third of the turkers mentioned the event of fastening the seat belts in the plane (BUCKLE_SEAT_BELT), nobody wrote about *undoing* their seat belts again, although in reality both events appear equally often. Consequently, we added an event type label for buckling up, but no label for undoing the seat belts.

### 4.2.3 First Annotation Phase

We used the WebAnno annotation tool (Yimam et al., 2013) for the annotation. The stories from each scenario were distributed among four different annotators, undergraduate students of computational linguistics. In a calibration phase, annotators were presented with some sample texts for test annotations; the results were discussed with the authors. Throughout the whole annotation phase, annotators could discuss any emerging issues with the authors. Due to the complexity of the task, we decided for a single annotation mode. To assess annotation quality, a small sample of texts was annotated by all four annotators and their inter-annotator agreement was measured (see Section 4.3.1).

Annotation of the corpus together with some pre- and post-processing of the data required approximately 500 hours of work. All stories were annotated with event and participant types (a total of 12,188 and 43,946 instances, respectively). On average there were 7 coreference chains per story with an average length of 6 tokens.

### 4.2.4 Modification of the Schema

Based on the results of the first annotation round, we extended and modified the templates. As already mentioned, we used MissScrEv and MissScrPart labels to mark verbs and noun phrases instantiating events and participants for which no appropriate labels were available in the templates. Based on the instances with these labels (a total of 941 and 1717 instances, respectively), we extended the guidelines to cover the most frequent cases.

In order to include new labels for event and participant types, we tried to estimate the number of instances that would fall under a certain label. We added new labels according to the following conditions:

- For the participant annotations, we added new labels for types that we expected to appear at least 10 times in total in at least 5 different stories (i.e. in approximately 5% of the stories).
- For the event annotations, we chose those new labels for event types that would appear in at least 5 different stories.

In order to avoid too fine a granularity of the templates, all other instances of MissScrEv and MissScrPart were re-labeled with ScrEv_other and ScrPart_other. We also relabeled participants and events from the first annotation phase with ScrEv_other and ScrPart_other, if they did not meet the frequency requirements. The event label air_bathroom (the event of letting fresh air into the room after the bath), for example, was only used once in the stories, so we relabeled that instance to ScrEv_other.

As mentioned above, we also looked at the DeScript corpus (Wanzare et al., 2016), which contains manually clustered event paraphrase sets for the 10 scenarios that are also covered by InScript (see Section 4.3.3). We extended our templates with additional labels for these events, if they were not yet part of the template.

### 4.2.5 Special Cases

***Noun-Noun Compounds.*** Noun-noun compounds were annotated twice with the same label (whole span plus the head noun), as indicated in Example 7. This redundant double annotation is motivated by potential processing requirements.

(7) I get my *(wash (cloth* $_{\text{ScrPart\_washing\_tools}}$*))*, $_{\text{ScrPart\_washing\_tools}}$ and put it under the water.

***Support Verb Complements.*** A special treatment was given to support verb constructions such as *take time*, *get home* or *take a seat* in Example 8. The semantics of the verb itself is highly underspecified in such constructions; the event type is largely dependent on the object NP. As shown in Example 8, we annotate the head verb with the event type described by the whole construction and label its object with SuppVComp (support verb complement), indicating that it does not have a proper reference.

(8) I step into the tub and *take*$_{\text{ScrEv\_sink\_water}}$ a *seat*$_{\text{SuppVComp}}$.

***Head of Partitive.*** We used the Head_of_Partitive label for the heads in partitive constructions, assuming that the only referential part of the construction is the complement. This is not exactly correct, since different partitive heads vary in their degree of concreteness (cf. Examples 9 and 10), but we could not see a way to make the distinction sufficiently transparent to the annotators.

(9) Our seats were at the *back*$_{\text{Head\_of\_Partitive}}$ of the train$_{\text{ScrPart_train}}$.

(10) In the library you can always find a *couple*$_{\text{Head\_of\_Partitive}}$ of interesting books$_{\text{ScrPart_book}}$.

***Mixed Participant Types.*** Group denoting NPs sometimes refer to groups whose members are instances of different participant types. In Example 11, the first-person plural pronoun refers to the group consisting of the passenger (*I*) and a non-participant (*my friend*). To avoid a proliferation of event type labels, we labeled these cases as Unclear.

(11) $I_{\text{ScrPart\_passenger}}$ wanted to visit *my*$_{\text{ScrPart\_passenger}}$ *friend*$_{\text{NPart}}$ in New York. ... *We*$_{\text{Unclear}}$ met at the train station.

We made an exception for the Getting a Haircut scenario, where the mixed participant group consisting of the hairdresser and the customer occurs very often, as in Example 12. Here, we introduced the additional ad-hoc participant label Scr_Part_hairdresser_customer.

(12) While *Susan*$_{\text{ScrPart\_hairdresser}}$ is cutting *my*$_{\text{ScrPart\_customer}}$ hair *we*$_{\text{Scr\_Part\_hairdresser\_customer}}$ usually talk a bit.

## 4.3 Data Analysis

| | Average Fleiss' Kappa | | | |
|---|---|---|---|---|
| | All Labels | | Script Labels | |
| Scenario | Events | Participants | Events | Participants |
| Bus | 0.68 | 0.74 | 0.76 | 0.74 |
| Cake | 0.61 | 0.76 | 0.64 | 0.75 |
| Flight | 0.65 | 0.70 | 0.62 | 0.69 |
| Grocery | 0.64 | 0.80 | 0.73 | 0.80 |
| Haircut | 0.64 | 0.84 | 0.67 | 0.86 |
| Tree | 0.59 | 0.76 | 0.63 | 0.76 |
| *Average* | *0.64* | *0.77* | *0.68* | *0.77* |

**(a)** Average Fleiss' Kappa.

| Scenario | %Coref Agreement |
|---|---|
| Bus | 88.9 |
| Cake | 94.7 |
| Flight | 93.6 |
| Grocery | 93.4 |
| Haircut | 94.3 |
| Tree | 78.3 |
| *Average* | *90.5* |

**(b)** Coreference agreement.

**Figure 4.3:** Inter-annotator agreement statistics.

### 4.3.1 Inter-Annotator Agreement

In order to calculate inter-annotator agreement, a total of 30 stories from 6 scenarios were randomly chosen for parallel annotation by all 4 annotators after the first annotation phase[2]. We checked the agreement on these data using Fleiss' Kappa (Fleiss, 1971). The results are shown in Figure 4.3a and indicate moderate to substantial agreement (Landis and Koch, 1977). Interestingly, if we calculate the Kappa only on the subset of cases that were annotated with script-specific event and participant labels by all annotators, results are better than those of the evaluation on all labeled instances (including also unrelated and related non-script events). This indicates one of the challenges of the annotation task: In many

[2]We did not test for inter-annotator agreement after the second phase, since we did not expect the agreement to change drastically due to the only slight changes in the annotation schema.

cases it is difficult to decide whether a particular event should be considered a central script event, or an event loosely related or unrelated to the script.

For the coreference chain annotation, we calculated the percentage of pairs which were annotated by at least 3 annotators (qualified majority vote) compared to the set of those pairs annotated by at least one person (see Figure 4.3b). We take the result of 90.5% between annotators to be a good agreement.

### 4.3.2 Annotated Corpus Statistics

Figure 4.4 gives an overview of the number of event and participant types provided in the templates. TAKING A FLIGHT and GETTING A HAIRCUT stand out with a large number of both event and participant types. In contrast, PLANTING A TREE and GOING ON A TRAIN contain the fewest labels. There are 19 event and participant types on average.

| Scenario | Events | Participants |
|---|---|---|
| BATH | 20 | 18 |
| BICYCLE | 16 | 16 |
| BUS | 17 | 17 |
| CAKE | 19 | 17 |
| FLIGHT | 29 | 26 |
| GROCERY | 19 | 18 |
| HAIRCUT | 26 | 24 |
| LIBRARY | 17 | 18 |
| TRAIN | 15 | 20 |
| TREE | 14 | 15 |
| *Average* | *19.2* | *18.9* |

**Figure 4.4:** The number of participants and events in the templates.

Figure 4.5 presents overview statistics about the usage of event labels, participant labels and coreference chain annotations. As can be seen, there are usually many more mentions of participants than events. For coreference chains, there are some chains that are really long (which also results in a large scenario-wise standard deviation). Usually, these chains describe the protagonist.

| | avg | min | max |
|---|---|---|---|
| event annotations in a story | 15.9 | 1 | 52 |
| event types in a story | 10.1 | 1 | 23 |
| participant annotations in a story | 52.3 | 16 | 164 |
| participant types in a story | 10.9 | 2 | 25 |
| coref chains | 7.3 | 0 | 23 |
| tokens per chain | 6 | 2 | 52 |

**Figure 4.5:** Annotation statistics over all scenarios.

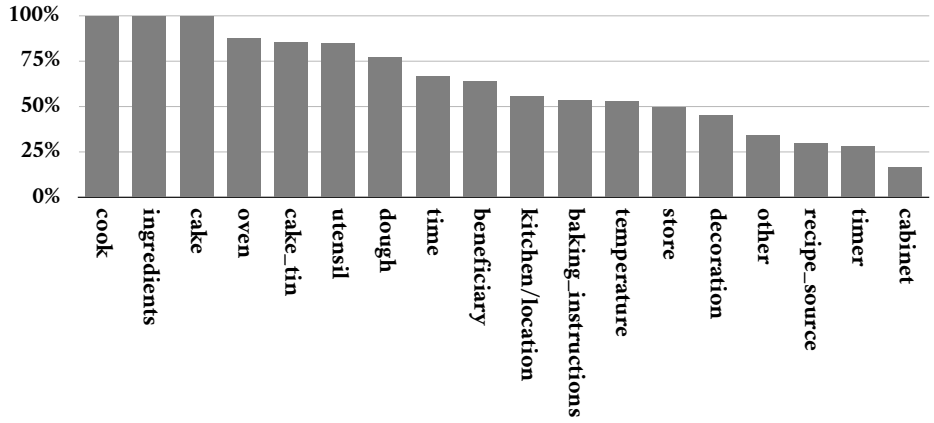



**Figure 4.6:** The percentage of stories in the BAKING A CAKE scenario that contain a certain participant label.

We also found that the FLYING IN AN AIRPLANE scenario stands out in terms of participant mentions, event mentions and average number of coreference chains.

Figure 4.6 shows for every participant label in the BAKING A CAKE scenario the number of stories which they occurred in. This indicates how relevant a participant is for the script. As can be seen, a small number of participants are highly prominent: COOK, INGREDIENTS and CAKE are mentioned in every story. The fact that the protagonist appears most often consistently holds for all other scenarios, where the acting person appears in every story, and is mentioned most frequently.

Figure 4.7 shows the distribution of participant/event type labels over all appearances over all scenarios on average. The groups stand for the most frequently appearing label, the top 2 to 5 labels in terms of frequency and the top 6 to 10. SCREV_OTHER and SCRPART_OTHER

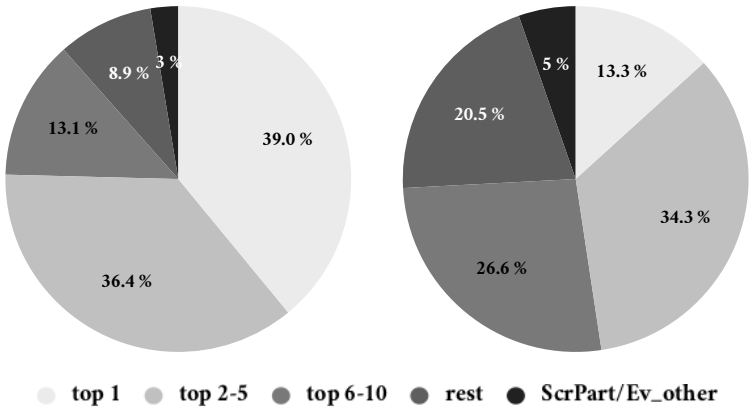



**Figure 4.7:** Distribution of participants (left) and events (right) for the first, the top 2 to 5 and the top 6 to 10 most frequently appearing events/participants, ScrEv/ScrPart_Other and the rest.

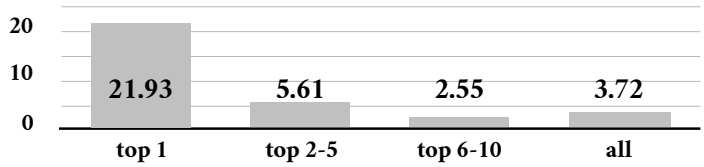



**Figure 4.8:** Average number of participant mentions for a story, for the first, the top 2 to 5 and the top 6 to 10 most frequently appearing events/participants, and the overall average.

are shown separately. As can be seen, the most frequently used participant label (the protagonist) makes up about 40% of the overall participant instances. The four labels that follow the protagonist in terms of frequency appear in 37% of the cases. More than 66% of all participant mentions refer to one of the top 5 labels.

In contrast, the distribution for events is more balanced. 13% of all event instances have the most prominent event type.

ScrEv_other and ScrPart_other both appear as labels in at most 5% of all event and participant instantiations: The specific event and participant type labels in our templates cover by far most of the instances.

In Figure 4.8, we grouped participants similarly into the first, the top 2 to 5 and the top 6 to 10 most frequently appearing participant types. The figure shows for each of these groups the average frequency per story, and in the rightmost column the overall average. The results

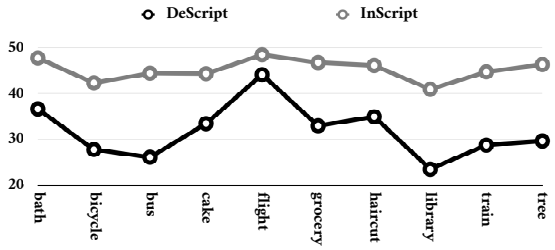



**Figure 4.9:** MTLD values for DeScript and InScript, per scenario.

correspond to the findings from the last paragraph.

### 4.3.3 Comparison to the DeScript Corpus

Since both InScript and DeScript will be used for the script parsing, it is interesting to compare both resources. The InScript corpus exhibits much more lexical variation than DeScript. Many approaches use the *type-token ratio* to measure this variance. However, this measure is known to be sensitive to text length (see e.g. Tweedie and Baayen (1998)), which would result in very small values for InScript and relatively large ones for DeScript, given the large average difference of text lengths between the corpora. Instead, we decided to use the *Measure of Textual Lexical Diversity (MTLD)* (McCarthy and Jarvis, 2010; McCarthy, 2005), which is familiar in corpus linguistics. This metric measures the average number of tokens in a text that are needed to retain a type-token ratio above a certain threshold. If the *MTLD* for a text is high, many tokens are needed to lower the type-token ratio under the threshold, so the text is lexically diverse. In contrast, a low *MTLD* indicates that only a few words are needed to make the type-token ratio drop, so the lexical diversity is smaller. We use the threshold of 0.71, which is proposed by the authors as a well-proven value.

Figure 4.9 compares the lexical diversity of both resources. As can be seen, the InScript corpus with its narrative texts is generally much more diverse than the DeScript corpus with its short event descriptions, across all scenarios. For both resources, the FLYING IN AN AIRPLANE scenario is most diverse (as was also indicated above by the mean word type overlap). However, the difference in the variation of lexical variance of scenarios is larger for DeScript than for InScript. Thus, the properties of a scenario apparently influence the lexical variance of the event descriptions more than the variance of the narrative texts.

We used entropy (Shannon, 1948) over lemmas to measure the variance of lexical realiza-

tions for events. We excluded events for which there were less than 10 occurrences in DeScript or InScript. Since there is only an event annotation for 50 ESDs per scenario in DeScript, we randomly sampled 50 texts from InScript for computing the entropy to make the numbers more comparable.

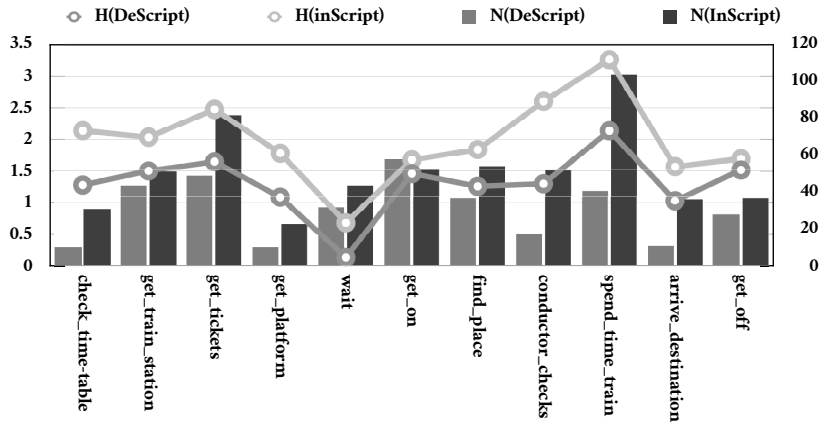



**Figure 4.10:** Entropy over verb lemmas for events (left y-axis, $H(x)$) in the GOING ON A TRAIN SCENARIO. Bars in the background indicate the absolute number of occurrence of instances (right y-axis, $N(x)$).

Figure 4.10 shows an example of the entropy values (abbreviated as *H*) for the event types in the GOING ON A TRAIN scenario. As can be seen in the graph, the entropy for InScript is in general higher than for DeScript. In the stories, a wider variety of verbs is used to describe events. There are also large differences between events: While WAIT has a very low entropy, SPEND_TIME_TRAIN has an extremely high entropy value. This event type covers many different activities such as reading, sleeping etc.

## 4.4 Conclusion

In this chapter, we described the InScript corpus, a corpus of narrative texts annotated with script structure and coreference information, which forms a unique resource for studying the role of script knowledge in NLP. We described the annotation process, various difficulties encountered during annotation and different remedies that were taken to overcome them.

In the following sections, we will use InScript as evaluation data for our script parser. In Chapter 5, we will first perform a thorough analysis and a further annotation of the data to find out about the linguistic complexity of the script parsing task. In Chapter 6, we will present our script parsing model that is trained on DeScript and evaluated on InScript.

Chapter 5

# Analysis of the Script Parsing Data

Since there has been no previous work on the task of script parsing, it is unclear how challenging the task actually is, and which kind of linguistic information and knowledge is required when aligning a text with a script. In this section, we try to estimate the difficulty of the script parsing task by means of an annotation study.

As in the rest of this thesis, we assume an underlying script structure in the form of paraphrase sets. Each paraphrase sets contains a number of short, telegram-style Event Descriptions (ED) that describe the event in simple words.

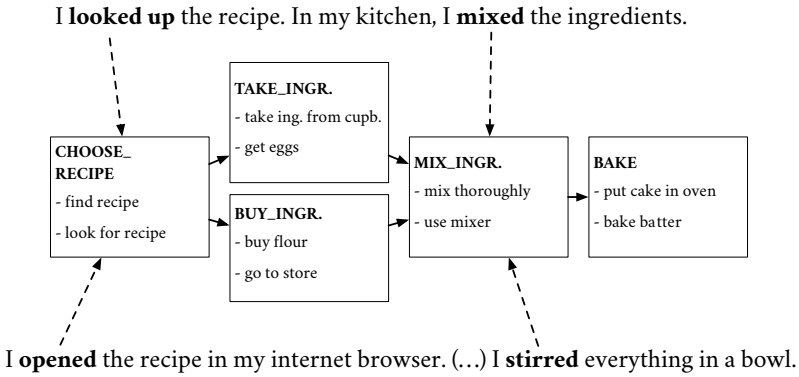



**Figure 5.1:** Aligning two text snippets of varying difficulty with the BAKING A CAKE script.

Consider the two example text fragments in Figure 5.1. Both the upper and the lower text fragment instantiate the CHOOSE_RECIPE and the MIX_INGREDIENTS events. Mapping the

event mentions in the upper part to the script is straightforward. Both verbs, *look* and *mix*, are mentioned in the respective paraphrase sets, so the mapping is trivial and just requires a simple word lookup. In contrast, the lower event mentions are more challenging. A script parsing model has to infer that *open the recipe* in this context describes the same event as the EDs in the paraphrase set, *find recipe* and *look for recipe*. Likewise, *stir* is a synonym of *mix*, which needs to be inferred to align the second event mention.

In this chapter, we investigate on a larger scale which types of knowledge and inference are required to assign correct event types to event mentions in text. We also want to find out to which degree crowdsourced script representations help to facilitate text-to-script mapping, i.e. how often the alignment is trivial, as in the upper part of Figure 5.1. We do so by annotating the type of semantic relations between pairs of an event mention and the paraphrase set representing the corresponding event type.

The structure of this chapter is as follows:

- We provide a manual annotation of word-level entailment types on verbs/events and nouns/participants (e.g. Synonymy, Hypernymy, Inference etc., Section 5.2) based on InScript and DeScript.
- In a second step, we compositionally derive clause-level entailment types (e.g. Equality, Entailment, etc., Section 5.3) that illustrate the types of inference that need to be modeled for the alignment of the clause with an event paraphrase set.
- In Section 5.4, we provide a detailed analysis of our annotation. We show that paraphrase sets as a constitutive part of script representations massively reduce the difficulty of automatic text-to-script mapping and increase its accuracy. We also find that a substantial sub-class of cases cannot be handled using just identity checks or shallow semantic modeling, but require deeper inference methods.

In the next section, we briefly introduce the datasets that were used for our annotation and describe an additional participant type annotation that was a prerequisite for the entailment annotations.

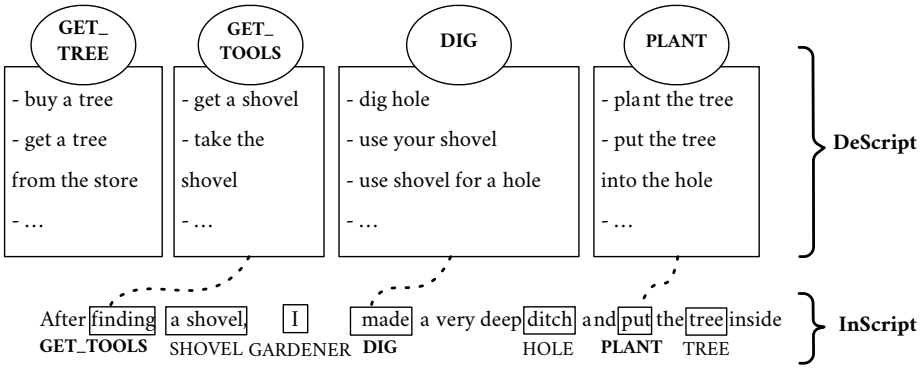



**Figure 5.2:** An example annotation in InScript (lower part), and the corresponding paraphrase sets from DeScript (upper part), for the PLANTING A TREE scenario.

## 5.1 Data

For our study, we use two existing resources that form the basis for evaluating text-to-script mapping: *InScript*, described in Chapter 4, and *DeScript* (Wanzare et al., 2016), a resource of structured script knowledge in the form of paraphrase sets, covering the same 10 scenarios. As can be seen in the upper part of Figure 5.2, each verb that is labeled in InScript has a corresponding paraphrase set in DeScript. This provides a gold standard alignment between textual event mentions and paraphrase sets, as indicated by the dotted lines, which is the basis for our annotation. The paraphrase sets in this kind of representation contain not only lexical synonyms, but also scenario-specific paraphrases of the same event: *use your shovel* and *dig hole* are not synonyms in a narrow sense, but in the context of PLANTING A TREE, they both describe the DIG event.

While DeScript provides manually created gold event paraphrase sets, the corresponding information on participant level is missing. Since we need entailment relations on the noun phrase/participant level for our compositional derivation of clause-level entailment (see Section 5.3), we extended DeScript with paraphrase sets for participant descriptions. For this purpose, we annotated all nouns in the 10 scenario subset of DeScript with labels from the InScript inventory of participant types. Data from the TAKING A BUS and PLANTING A TREE scenarios were annotated by two annotators to assess the inter-annotator agreement, which was *almost perfect* (Landis and Koch (1977), $\kappa = 0.91$).

From this annotation, paraphrase sets for participants were derived, i.e. the sets of all nouns

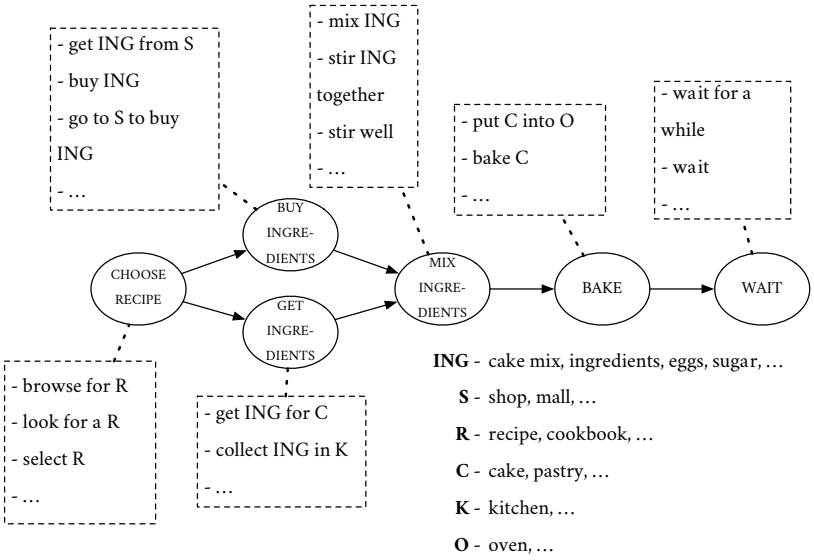



**Figure 5.3:** A full script with participant paraphrase sets.

describing the same participant. In the BAKING A CAKE scenario for example, all different ingredients such as eggs, flour, milk etc. are members of the INGREDIENT paraphrase set. Figure 5.3 shows a full script for the BAKING A CAKE scenario, with participant paraphrase sets in the lower right corner.

For our study, we selected 3 out of the 10 scenarios that differ with respect to their complexity: TAKING THE BUS, BAKING A CAKE and PLANTING A TREE. We annotated script-relevant verb instances and participant instances in every story of these 3 scenarios.

## 5.2 Lexical Entailment: Annotation Study

To identify the type of semantic relations that need to be modeled in order to align an event-denoting clause in the text with a paraphrase set representing the same event, the most straightforward way would be to conduct a clause-level entailment annotation between the clause and EDs in the paraphrase set, e.g. with a set of clausal entailment types similar to the ones used in MacCartney and Manning (2009).

We found, however, that the assessment of entailment types is time-consuming and unreliable, when based on a direct comparison of complex text clauses and paraphrase sets. We therefore simplified the task, breaking it down into two steps: First, annotators were asked to assign semantic relations between the event descriptions in the paraphrase sets and their

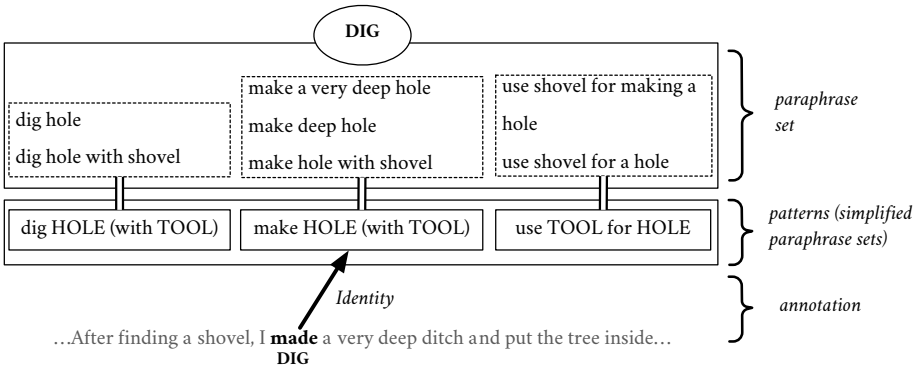



**Figure 5.4:** Combining the EDs in the DIG event into patterns (upper part) and the actual verb annotation (lower part).

instantiations in narrative texts *on the lexical level only*, labeling the event-denoting verbs and the participant-denoting noun phrases, as described in this section. Second, we automatically derive an approximate clause-level entailment type from the lexical-level labels in a quasi-compositional way from the manually annotated lexical entailment relations, as addressed in Section 5.3.

In the following subsections, we describe the lexical entailment annotation we conducted on the DeScript and InScript data for events/verbs (Section 5.2.1) and participants/nouns (Section 5.2.2).[1] Annotation guidelines are given in the appendix, in Sections C and D.

### 5.2.1 Events

To prepare the data for the event annotation, we made use of the gold alignment (cf. Figure 5.2): Each event-denoting verb in InScript was presented with the corresponding paraphrase set in DeScript.

Comparing *every* ED in the paraphrase set to the clause in the text is cumbersome due to the number of EDs (up to 76) in each paraphrase set. We therefore simplified the event paraphrase sets by building equivalence classes of EDs that use the same head verb, which we call *event patterns*. They are derived semi-automatically by summarizing EDs that contain the same main verb, and replacing noun phrases with their participant type label (upper part

[1] Due to a technical error, a small part of the data were not annotated (approx. $4-5\%$). Because the original annotators were not available anymore, we decided to not annotate these instances to avoid inconsistencies. A manual inspection of the instances revealed no systematic differences to the rest of the data.

of Figure 5.4). Also, participants that do not appear in every ED are marked as optional (with brackets).

In the annotation process, annotators were only shown the patterns instead of the full paraphrase set. In the lower part of Figure 5.4, the verb labeled as DIG is compared to the event patterns extracted from the DIG paraphrase set. Verbs were presented in their sentential context and highlighted.

Instead of annotating each pattern, the guidelines required annotators to select only the most similar pattern for the event-denoting verb, and to do the annotation only for this pattern. While this selection results in a non-exhaustive annotation, no important information is lost: The procedure of selecting the most similar pattern retains the minimal inference steps required for the alignment.

After selecting a pattern, annotators were instructed to assess the relation between the verb in DeScript and the verb in InScript and use the context only for lexical disambiguation, i.e. not to assess clause-level entailment. In our annotation schema, we include the following labels:

- *Identity, Synonymy, Hyponymy, Hypernymy.* Defined as in *WordNet* (Fellbaum, 1998). *Hyponymy* describes the case in which the verb in the text is more specific than in the pattern, *Hypernymy* is the opposite.

- *Incorporation.* One verb includes a participant, which is explicitly mentioned with the other verb.
  Example: I ***sprinkled flour*** *in the pan.* – ***flour*** CAKE_TIN

- *Diathesis.* This covers active/passive alternation (Ex. 1) and verbs that are conceptually equivalent but have different syntactic realizations. In *FrameNet* (Ruppenhofer et al., 2006), these verbs would typically be associated with the same frame (Ex. 2).
  Example 1: *The cake* ***was cut.*** *–* ***cut*** CAKE
  Example 2: *The cake* ***went*** *in the preheated oven –* ***put*** *(*CAKE*) in* OVEN

- *Phrasal Verb.* One of the verbs is a particle verb that has the same meaning as the other verb.
  Example: *I* ***went out*** *to the grocery store.* – ***go*** *to* STORE

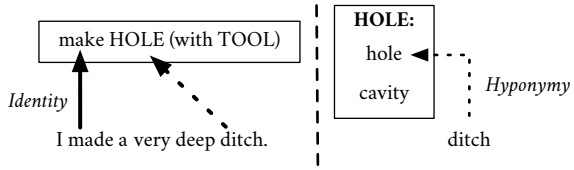



**Figure 5.5:** Aligning participants with a pattern (left), selecting a noun from the participant paraphrase set and labeling (right).

- *Inference.* A complex inference is needed to associate the verbs with the same event[2].
  Example 1: *I **put** the decorations on the cake. - **use** DECORATION*
  Example 2: *I **made sure** that I had all the ingredients that I needed. - **gather** INGREDIENTS*
- *NoMatch.* No match is possible. This is typically the case when annotation errors occurred in InScript.

### 5.2.2 Participants

Associating verbs in a textual mention with the ED verb is straightforward, since there is only one head verb on each side. Linking participants in contrast requires additional annotation effort: They can be left out or appear more than once in one clause. We thus divided the participant annotation into two steps:

**Participant Alignment.** In order to distinguish missing from realized participants, annotators had to align all participants that are relevant to the event in question in the text with their counterparts in the pattern, i.e. with the participant type labels. In Figure 5.5, this step corresponds to the dotted arrow on the left hand side. Participants that were mentioned in the text and that play a role in the event, but that do not have a counterpart in the ED, were marked as *Additional*. Required participants in the pattern that were not aligned were afterwards automatically labeled as *Missing*.[3] As an example for this label, consider the text clause *I used a shovel*, with the matching pattern *use TOOL for HOLE*. The participant HOLE is omitted, but mentioned as mandatory in the pattern, i.e. it appears in every ED of the

[2] In the original guidelines, there were two labels indicating a complex inference: *Context-Relatedness* for a weaker type of inference, and *Inference* for the rest. During the annotation, we found that both labels were used interchangeably and inconsistently, so we decided to merge them.

[3] The protagonist of the story, being very rarely mentioned in the EDs, was excluded from the alignment.

respective paraphrase set (*use shovel for making a* ***hole*** and *use shovel for* ***hole***). Therefore, it is marked as *Missing* in the pattern[4].

**Lexical Entailment Annotation.** To find the appropriate type of lexical entailment for the realized participants, annotators were shown the participant paraphrase sets from DeScript for all aligned participants. Just as for event patterns, annotators had to choose the best matching, most similar noun from the set and assign a lexical entailment label.

As for verbs, the annotation schema includes the relation types, *Identity, Synonymy, Hyponymy, Hypernymy, Meronymy* and *Holonymy* and the additional labels *Co-Hyponomy*, *Inference* and *NoMatch*, as defined in Section 5.2.1. We add the label *Instance*, which is used when the noun in InScript is a proper noun or entity mention of the type expressed by the DeScript noun (e.g. *number 77 – bus*).

The right hand side of Figure 5.5 illustrates this part of the annotation: The noun "ditch" (which was previously aligned with the pattern) is compared to all participant descriptions for HOLE, linked to the lexical description "hole" and labeled with *Hyponymy*. Figure 5.8 shows the fully labeled instance with participant and event annotations.

To simplify the annotation, we make the assumption that each noun has only one sense per scenario: In the PLANTING A TREE scenario, the polysemous word *stem* e.g. always describes a part of a tree. In order to reduce the annotation effort, we presented all different noun types per participant type only once, rather than every single mentioned token in its sentential context. This *on sense per scenario* assumption is similar to the *one sense per discourse* hypothesis, which is often used in word sense disambiguation models (Gale et al., 1992).

## 5.3 Clause-Level Entailment: Composition

In the previous section, we described the lexical entailment annotation on verbs and nouns, i.e. on a sub-event level. In this section, we now explain a method for an automatic, quasi-compositional computation of clausal-level entailment types. We compose the types from the lexical-level entailment labels of the verb and all its annotated noun dependents.

[4]These labels correspond to the participant labels listed in C. The labels *Shared Participant* and *Label Inconsistency* were not used by the annotators and ignored.

| Entailment Type | Labels |
|---|---|
| Identity $=$ | Identity |
| Equality $\equiv$ | Synonymy, Phrasal Verb, Diathesis |
| Entailment $\sqsubset$ | Hyponymy, Instance, *Additional* |
| Reverse ent. $\sqsupset$ | Hypernymy, *Missing* |
| Partial Entailment $\infty$ | Inference, Incorporation, Meronymy, Holonymy, Co-Hyponymy |
| Non-Entailment # | NoMatch |

**Figure 5.6:** Entailment types.

Inspired by the textual inference method used in MacCartney and Manning (2007) and MacCartney and Manning (2009), we compute the type of clause-level entailment between InScript event mentions and DeScript patterns from the manually annotated word-level entailment labels. Following MacCartney, we group these labels according to their truth-conditional effects, and associate each group with one of six entailment types, shown in Figure 5.6. We adopt four entailment types from the schema of MacCartney and Manning (2009) and add two new types: We extend the schema with *Identity*, which is logically speaking a sub-case of *Equality*. We also use *Partial Entailment* to cover all cases of semantic relatedness which do not correspond to a direct entailment type; most prominent are the *Inference* cases. Cases of *Additional* and *Missing* participants are deletions and insertions in MacCartney's terminology, and therefore have entailment and reverse entailment effects, respectively.

To compute the clausal entailment type, we combine the word-level entailment labels for verbs and nouns, according to the composition table in Figure 5.7. The result of combining two lexical labels is in general the weaker entailment relation of the two[5]. The only exception is the pair $\{\sqsubset, \sqsupset\}$, which results in *Partial Entailment*. The application of the binary computation is commutative, so the clausal entailment type can be read off the set of lexical labels[6].

[5] The list $=, \equiv, \{\sqsubset, \sqsupset\}, \infty, \#$ orders the labels from strongest to weakest.

[6] Our composition step disregards non-affirmative verbs or other words with a negative polarity, such as *avoid* or *prohibit*, which would reverse the composed entailment type. Due to the simple language and straight-

| | $=$ | $\equiv$ | $\sqsubset$ | $\sqsupset$ | $\infty$ | # |
|---|---|---|---|---|---|---|
| $=$ | $=$ | $\equiv$ | $\sqsubset$ | $\sqsupset$ | $\infty$ | # |
| $\equiv$ | $\equiv$ | $\equiv$ | $\sqsubset$ | $\sqsupset$ | $\infty$ | # |
| $\sqsubset$ | $\sqsubset$ | $\sqsubset$ | $\sqsubset$ | $\infty$ | $\infty$ | # |
| $\sqsupset$ | $\sqsupset$ | $\sqsupset$ | $\infty$ | $\sqsupset$ | $\infty$ | # |
| $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | # |
| # | # | # | # | # | # | # |

**Figure 5.7:** Combination table for lexical entailment classes.

Figure 5.8 shows the running example with all lexical entailment annotations. The set of word-level entailment labels is $\{=, \sqsubset\}$, given the lexical entailment labels {*Identity, Hyponymy*}. The clausal entailment type is the weakest lexical type, i.e., $\sqsubset$ (*Reverse Entailment*).

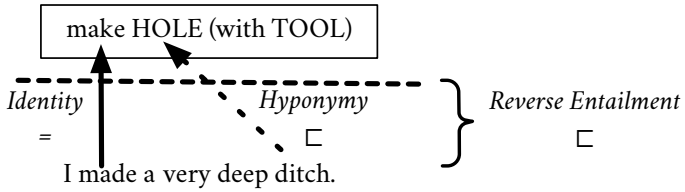



**Figure 5.8:** One fully labeled event instance, with compositionally derived clausal entailment label (*Reverse Entailment*).

## 5.4 Annotation Statistics

The annotation was performed on all 280 texts of the InScript corpus addressing one of the three selected scenarios. Annotation was done by two native speakers of German with a good command of English. A total of 3,427 verb mentions and 1,248 noun types were annotated, respectively[7]. We used SWAN[8] Gühring et al. (2016) for the annotation.

forward formulations in InScript, such expressions however occur only rarely, if ever.

[7] Event verbs of type ScrEv_other were excluded from the annotation, since they have no consistent counterpart in DeScript.

[8] https://github.com/annefried/swan

| Label | Events | Participants |
|---|---|---|
| Identity | 58% | 76% |
| Synonymy | 5% | 6% |
| Hyponymy | 5% | 5% |
| Phrasal Verb | 5% | - |
| Inference | 13% | 1% |
| NoMatch | 7% | 7% |
| Other | 7% | 5% |

**Table 5.1:** Distribution of lexical labels on events and participants.

### 5.4.1 Lexical Level

**Inter-Annotator Agreement** For both verb and participant annotation, agreement is computed on two levels: First, we report how often the same pattern or head noun was selected. Second, in cases where the same pattern/noun was selected, we report the label agreement. For the verb annotation, annotators chose the same pattern in 82.4% of cases. For participants, the annotators chose the same realization from DeScript in 74.3% of cases.

On verbs, the annotators agreed on the label with $\kappa = 0.72$ (*substantial agreement*, Landis and Koch (1977)). On participants, they agreed with $\kappa = 0.742$ (*substantial agreement*).

Not every case of disagreement is critical: In many cases there is more than one plausible solution. In the PLANTING A TREE scenario, for example, the word *shovel* could be interpreted either as a *Synonym* or *Co-Hyponym* of *spade*. There are also no sharp boundaries between classes. In particular, annotators had difficulties with annotating *Inference* cases. Therefore, we decided not to adjudicate the annotation, but average over the distributions for evaluation.

**Label Distribution** Table 5.1 gives label distributions for participants and events on the lexical level, averaged over both annotations. As mentioned before, we annotated each noun type only once per participant type rather than annotating every mention of a noun separately. To compute the numbers depicted in Table 5.1, we transfer the type-level annotation

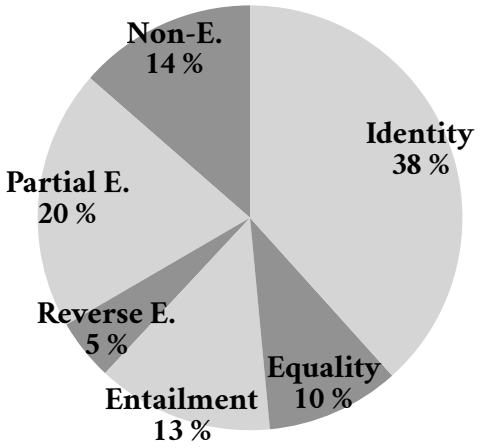



**Figure 5.9:** Distribution of composed labels.

to every single appearance of the noun, to give a better idea of the actual distribution.

For both verbs and participants, the most frequent relation chosen by both annotators is *Identity*. Among the lexical relations, *Hyponymy* is more frequent than *Hypernymy*, which is consistent with the expectation that concrete event/participant mentions use more specific verbs/nouns than the abstract descriptions in the script knowledge base (cf. Modi (2016)). *Diathesis*, *Incorporation* and *Hypernymy* for verbs, and *Meronymy*, *Hypernymy*, *Holonymy*, *Instance* and *Co-Hyponymy* for participants appear only very rarely and are subsumed under *Other* in the table.

### 5.4.2 Clausal Level

Figure 5.9 shows the distribution of the resulting clausal entailment labels for both annotators. *Identity* makes up for the largest part of cases (38 %), illustrating the high lexical coverage of crowdsourced script representations.

*Entailment* cases are significantly more frequent than *Reverse Entailment*, which is in line with the leading assumption that an event mention should entail a description of its event type, and with the observation that event-denoting clauses usually use longer sentences and more specific vocabulary than event descriptions given by the script knowledge base.

Both *Entailment* and *Reverse Entailment* mainly contain cases in which the participant is not realized on one side, and are only rarely composed from *Hypernymy* or *Hyponymy* cases. A

typical example from the TAKING A BUS scenario is the text clause *wait for several minutes*. The most similar ED in the scenario is just *wait*, so the time expression is an additional participant and thus results in a clause-level entailment.

There is a large number of *Partial Entailment* cases (20%), which are in many cases composed of one or several *Inference* labels. One typical example of such a case from the TAKING A BUS scenario is the text clause *the driver pulled over*. The paraphrase sets only contain phrases like *bus stops* or *arrive at destination*. In this case, a system would need to know that *pull over* in the bus context is a paraphrase for *stop*, which requires contextual inference. These cases also appear in other scenarios, e.g. in the PLANTING A TREE scenario: *look at trees* is a contextual paraphrase of *choose a tree*. This can be seen as an indicator for the difficulty of the text-to-script mapping task, but it also indicates that the script resource is not exhaustive enough to contain all possible formulation variants for events.

Lastly, we found that a large number of *Non-Entailment* cases are derived from annotation errors in InScript.

## 5.5 Conclusion

In this chapter, we annotated event mentions in narrative texts with semantic relations that need to be modeled when mapping the mentions to script events that are represented as paraphrase sets. We provided a lexical-level entailment annotation between event-denoting verbs and participant-denoting nouns of narrative texts on the one side, and event and participant descriptions of a script on the other side. We then derived clause-level entailment labels that highlight the coverage of crowdsourced paraphrase sets associated with event types, as compared to the textual variation in naturalistic texts.

Our most important finding is that script representations in the form of paraphrase sets can cover a large number of description variants of an event in a text, which is indicated by the large number of *Identity* cases in the data. However, the alignment of a substantial amount of event mentions requires a deeper inference of multiple semantic relations.

Chapter 6

# A Script Parsing Model

In this chapter, we describe our script parsing model. This forms a central part of this dissertation: Our model is the first scalable script parser, that can be trained on arbitrary event sequences from a limited set of scenarios. We implement the script parser by fitting a linear-chain conditional random field on sequences of event descriptions. The model implicitly captures typical ordering information about events by learning a sequential feature.

As before, we use the DeScript dataset (Wanzare et al., 2016) as a script knowledge database. To evaluate our script parser, we use texts from the InScript collection (s. Chapter 4). InScript can be used as a gold standard for script parsing due to the annotated verbs: Every event-denoting verb in the texts in InScript can be associated with an event paraphrase set in DeScript.

Our script parsing model is trained only on ESDs, i.e. the InScript gold standard is used for evaluation purposes only. This makes sense, since ESDs are cheap and easy to acquire via crowdsourcing, while a full alignment annotation is highly complex (as seen in Chapter 4). The fact that the genres of training and test set differ between the bullet-point style ESDs and the naturalistic texts in InScript poses a challenge, but the model still shows convincing results.

Script parsing is a complex task, as stated in Chapter 3. We assumed that it consists of 5 subtasks: (1) Segmenting the text, (2) identifying segment scenarios, (3) identifying event

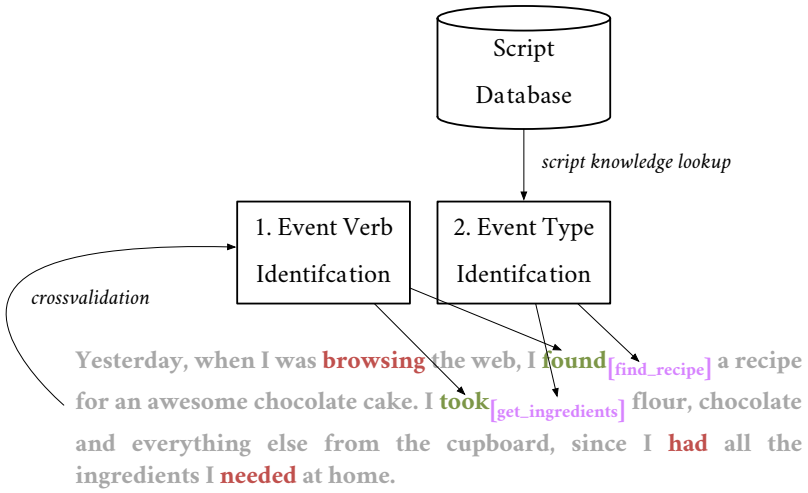



**Figure 6.1:** Structure of the proposed parser.

verbs, (4) identifying event types, and (5) identifying participants. However, writing a script parser that is generally applicable on every kind of text and that implements all these steps would go beyond the scope of this thesis, so we simplified the task:

- First, using the InScript corpus eliminates the need for text segmentation: Each text talks about one single scenario only.
- Second, InScript contains only texts for 10 different scenarios. The scenario identification is a trivial task, since the 10 used scenarios differ topically, and identifying the relevant script of a text comes down to spotting one or two key words. For the TAKING A TRAIN scenario, such key words may be *train* or *ticket*; for the TAKING A BATH scenario, key words are e.g. *bath* or *soap*. We therefore skip the scenario identification step and only perform a simple lookup to associate texts with the scenario.
- Lastly, our model only performs script parsing on events. We shall leave participant labeling for future work.

Consequently, our script parser consists of two modules corresponding to the remaining two tasks: The first module is an event verb identification module, which identifies verbs that denote script events of the given script. The second module is the event type identification module, the central component of the system. It identifies event types of verbs, i.e. it carries out the actual alignment task.

Figure 6.1 shows the structure of the model proposed in this section.

The structure of this chapter is as follows:

- In Section 6.2, we describe the first scalable script parsing model. It is based on conditional random fields for the identification of script event types on verbs in a text. The model is trained purely on ESD collections and can thus be scaled up to an arbitrary number of scenarios.
- In Section 6.1, we describe a model to identify event-denoting verbs in a text, which is trained in a cross-validation setting. We outline the importance of scenario and frame information and show that both types of information are crucial for identifying event mentions in texts.
- We evaluate both models separately. We also combine the models by integrating the event type identification after the event verb identification model and provide an end-to-end evaluation. We demonstrate that ordering information is in general beneficial for script parsing (Sections 6.3 and 6.4).
- On basis of the annotations from Chapter 5, we conduct a fine-grained evaluation which is described in Section 6.4.2. We show that the parser greatly benefits from the availability of many formulation variants, emphasizing the need for linguistically rich script databases. We also show that a substantial amount of alignment cases requires more complex inference steps.

## 6.1 Event Verb Identification

We make another technical simplification that affects the event verb identification step. As mentioned in Chapter 3, we assume that a script parser is only trained on script data that are acquired via crowdsourcing.

However, DeScript does not allow us to train a supervised event verb identification model. Due to the design of the data collection process of DeScript, it contains only descriptions of events of the scenario in question. A supervised event verb identification model however needs both negative and positive cases: Together with descriptions of events that belong to

the script scenario, it needs to have access to descriptions of non-events or events that do not belong to the scenario.

We tried unsupervised approaches, rule-based systems and several methods to artificially generate non-events for training, but found that all trials resulted in bad performance. Instead, we use InScript directly to learn the event verb identification model, since it contains both script-relevant verbs and non-script relevant verbs. The event verb identification model is then trained and evaluated on InScript via cross-validation.

We use a decision tree classifier for identifying script-relevant verbs (*J48* from the Weka toolkit, (Frank et al., 2016)). As presented in Chapter 4, annotations on InScript contain four classes of event type labels: A class for verbs that denote specific script events (i.e. scenario-specific labels plus ScrEv_Other), Evoking for script-evoking elements, RelNScrEv for related events that are not part of the script, and UnrelEv for verbs that denote events that are too general or not relevant to the current script, as well as verbs that do not denote events at all.

At test time, we merge Evoking, RelNScrEv and UnrelEv into one *non-script event* class, since these three classes all contain verbs that do not denote events of the current script. At training time, in contrast, the classifier takes into account all four classes, since we found this to be helpful for the performance.

We use the following feature types:

- **Syntactic Features.** To detect verbs that do not denote script events at all (i.e. a part of the UnrelEv class), we employ syntactic features: a feature for auxiliaries; for verbs that govern an adverbial phrase (mostly if-clauses); a feature indicating the number of direct and indirect objects; and a lexical feature that checks if the verb belongs to a predefined list of non-action verbs, taken from an online lexicon for English learners.[1].

- **Script Features.** For finding verbs that describe only very general events (such as *decide*), or for verbs that do not denote events of the current script (i.e. the UnrelEv and RelNScrEv classes), we employ a set of script features: a binary feature indicat-

[1] *be, seem, appear, look, sound, smell, taste, feel, like, want, prefer, love, have, own, possess, think, believe, consider*, from https://eslgold.com/grammar/nonaction_verbs/

ing whether the verb is used in the ESDs for the given scenario; and a scenario-specific tf–idf score that is computed by treating all ESDs from a scenario as one document, summed over the verb and its dependents. In Section 6.3.2, we compare models with and without script features, to test the impact of scenario-specific information.

- **Frame Feature.** We further employ frame-semantic information. Frame information can on the one hand help to identify verbs that do not denote events at all: the *Feeling* frame will for example rarely denote a script event. On the other hand, it should help to abstract away from different event realizations within a scenario.

  We use a state-of-the-art semantic role labeller (Roth and Lapata, 2016) based on *FrameNet* (Ruppenhofer et al., 2006) to predict frames for all verbs, encoding the frame as a feature. We address sparsity of too specific frames by mapping frames to higher-level super frames using the *framenet querying package*[2].

We also assumed that there is a relation between the aspectual verb class and whether it describes a script event or not. We thus performed alternative experiments with a verb aspect classifier (Friedrich et al., 2016). This classifier assigns situation entity types to verb instances, containing categories such as *state* (a verb that describes a property), *generic* (a statement about the property of a category) or *event* (an actual action that happened). The vast majority of verbs in InScript were classified as *event* or *state*. Most verbs labeled as *event* were indeed verbs that instantiate script events, but there were also many script event verbs that were classified as *state* verbs, such as *fill* in Example 13:

(13) I let the tub fill up to about a hand 's height from the top.

Features computed based on verb aspect were not found to be helpful and excluded from further experiments.

## 6.2 Event Type Identification

In this section, we describe the core of our script parsing system, the event type identification model. We start out with a baseline model that is solely based on textual similarity. We then

[2] github.com/icsi-berkeley/framenet

describe a model that uses the temporal structure of a script, by fitting a sequence labeling model on sequences of script events.

### 6.2.1 Baseline: Similarity-Based Model

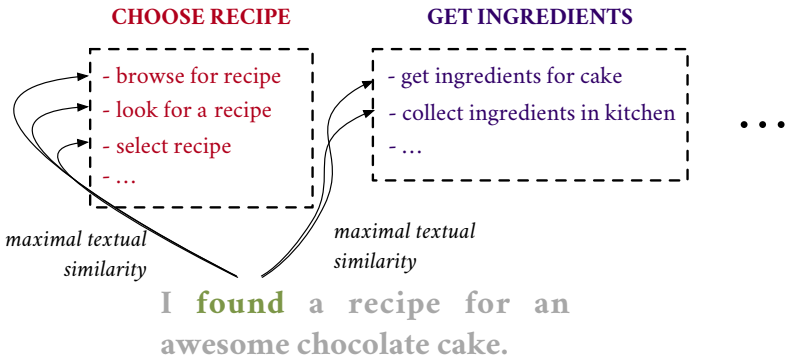



**Figure 6.2:** An event type identification model based on textual similarity.

Figure 6.2 shows a simple model that employs textual similarity for the event type identification step. The idea is to use a textual similarity measure to compare the event mention in the text with all EDs in the script knowledge resource. Then, the event type label of the paraphrase set with the most similar ED is assigned. We also tried selecting the event type label of the paraphrase set with the highest average similarity, but found this to perform worse. This was most likely due to the fact that in most paraphrase cluster, there is one ED that is mentioned very often. Computing the average biases models towards this most common ED, such that less common formulation variants are no longer recognized.

We use two similarity measures. The first measure is based on lemma equality. Let $T$ denote the lemmas of the event-denoting verb in the text together with all its dependency children. Let $E$ denote the lemmas of the verb and its dependency children of a event description. The similarity is defined as the average of (1) the proportion of lemmas in the textual event mention that are also mentioned in the ED and (2), vice versa, the proportion of lemmas in the ED that are also mentioned in the textual event mention:

$$sim_{lemma}(T, E) = \frac{\frac{|T \cap E|}{|T|} + \frac{|T \cap E|}{|E|}}{2} \tag{6.1}$$

The second similarity measure we use is based on *word2vec* word embeddings (Mikolov et al., 2013a,b). Let $\mathbf{t}$ be the centroid of the word vectors of lemmas in $T$, and let $\mathbf{e}$ be the centroid of

the word vectors of lemmas in $E$[3]. The word2vec similarity is defined as the cosine similarity of **t** and **e**:

$$sim_{w2v}(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{t} \cdot \mathbf{e}}{||\mathbf{t}||_2 ||\mathbf{e}||_2} \tag{6.2}$$

We also experimented with other similarity measures based on WordNet (Fellbaum, 1998), such as the Lin (Lin, 1998) or Resnik (Resnik, 1995) similarity. We found the word2vec similarity to generally outperform WordNet-based measures.

### 6.2.2 Sequence Labeling Model

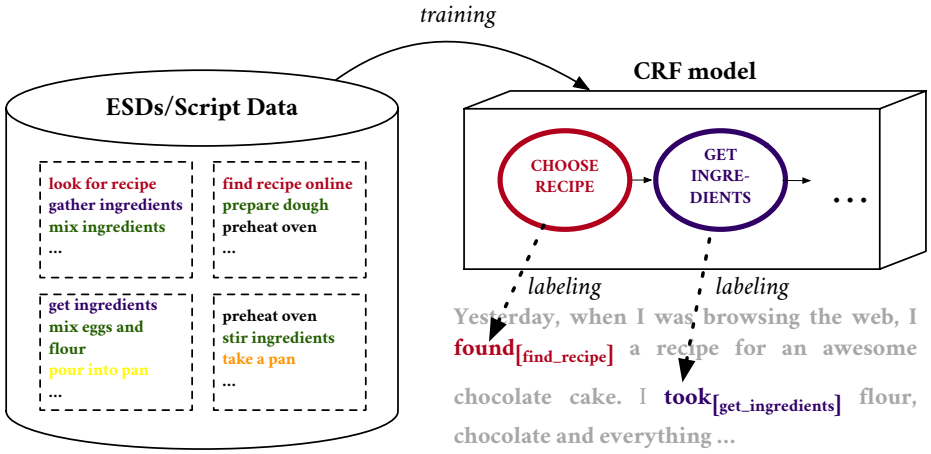



**Figure 6.3:** A CRF-based script parser.

One of the most important aspects of script knowledge is knowledge about the typical temporal order of events in an everyday situation. The textual similarity models proposed in the last section do not make use of such temporal information. In order to capture temporal order, the second model we propose is based on a conditional random field (CRF, Lafferty et al. (2001)). A CRF is a discriminative model that is used for sequence labeling tasks. By design, it makes use of ordering information, by considering a chain of observations and labels rather than a single observation/label.

In our case, the task is to predict a chain of event type labels on a narrative text. The model is trained on ESDs only, as shown in Figure 6.3. Each single training instance, i.e. each ESD,

[3]To emphasize the importance of the verb, its word vector is counted twice when computing the centroid.

consists of a sequence of single EDs (the *observations*) with respective event type labels (the *labels*). During test/prediction time, the CRF outputs event type labels on the event-relevant verbs in a narrative text. The observations thus correspond to the script-relevant verbs in the text at prediction time.

In a CRF, the probability of a label sequence $\overrightarrow{y}$ of length $T$ given an observation sequence $\overrightarrow{x}$, $P(\overrightarrow{y}|\overrightarrow{x})$ is defined as follows:

$$\frac{1}{Z(\overrightarrow{x})}\prod_{t=1}^{T} exp\left\{\sum_{i=1}^{m}\lambda_i f_i(y_{t-1}, y_t, x_t)\right\} \tag{6.3}$$

Here, $f_i ... f_m$ are feature functions for a given combination of an observation $x_t$, which is labeled as $y_t$, and the previous label $y_{t-1}$. $Z(\overrightarrow{x})$ is a normalization factor, and $\lambda_i$ is the weight of the feature function $f_i$ that is learned during training. In a CRF, feature functions are usually indicative, i.e. they return 1 if a given feature combination is in the data.

In our case, the observations $x_t$ during training are the single EDs of one ESD. The labels $y_t$ correspond to event type labels. The feature function in this case only considers the previous label, although in general, there is no such restriction for CRFs. This kind of CRF is also referred to as *linear-chain CRF*. The fact that the feature functions can access the previous label means that the model has access to context information - in our case, information about the previous event.

### Feature Types

For identifying the correct event type given a script-relevant verb, we leverage two types of information: We require a representation for the meaning and content of the event mention, which takes into account not only the verb, but also the persons and objects involved in an event, i.e. the *script participants*. In addition, we take event ordering information into account, which helps to disambiguate event mentions based on their local context.

Our implementation is based on the CRF++ toolkit[4]. In CRF++, *feature categories* are defined rather than single feature functions[5]. In simple words, feature categories are sets of similar feature functions. An example feature category for an ED would be the *verb lemma*. From

[4] taku910.github.io/crfpp/

[5] In CRF++, feature categories are just referred to as *features*.

the feature category *verb lemma*, the toolkit computes all possible feature functions: For each verb lemma in the training data, CRF++ compiles a feature function that returns 1 if the verb lemma is in the ED and 0 if not.

We employ two types of feature categories:

*Sequential Feature Category.* Our CRF model utilizes event ordering information in the form of binary indicator features that encode the co-occurrence of two event type labels in sequence.

*Meaning Representation Feature Categories.* Two feature categories encode the meaning of a textual event mention. One is a shallow form of representation derived from precomputed word embeddings (*word2vec*, (Mikolov et al., 2013a,b)). This feature category captures distributional information of the verb and its direct nominal dependents[6], which we assume to denote script participants, and is computed by averaging over the respective word vector representations, as for the computation of $sim_{w2v}$.

As a more explicit but sparse form of content representation, we additionally use the lemma of the verb, its indirect object and its direct object as feature categories.

### 6.2.3 Alternative: Hidden Markov Model

An alternative to using a CRF would be to employ a Hidden Markov Model (HMM). Unlike a CRF, a HMM is a generative sequence labeling model that models the joint probability of observation and label sequence ($P(\overrightarrow{y}, \overrightarrow{x})$). This probability is defined as:

$$p(y_0) \prod_{t=1}^{T} p(y_t|y_{t-1})p(x_t|y_t) \tag{6.4}$$

Here, $p(y_i|y_{i-1})$ is the transition probability of moving from label $y_{i-1}$ to $y_i$, and $p(x_i|y_i)$ is the *emission probability* that $y_i$ generates the observation $x_i$.

Notably, probabilities are usually not estimated via feature functions. Transition and emission probabilities can be read off a database in a supervised fashion, or estimated with an expectation maximization (EM) algorithm. On the script data that are used, this step is straightforward for the transition probabilities, since event label sequences can be read off the ESDs. For the emission probabilities, HMMs usually assume a fixed set of observations,

[6] For EDs, we use all mentioned head nouns.

which is a problem for a script parser: The number of phrases that can describe a specific event is arbitrarily large. We thus tried to approximate the emission probabilities with textual similarity measures (as for the similarity model), which would be normalized over all events.

We found that such an HMM model performs worse than a similarity-based model. One of the main reasons for this is that the transition probabilities are too restrictive. The script database we use is relatively small, with only 50 sequences per scenario. This means that there are plausible event sequences that are not covered. Such sequences would get a total probability of 0 in the HMM, given that one of the multipliers is 0[7].

Together with the fact that HMMs don't allow richer features representations, we found CRFs to perform better by a large margin.

## 6.3 Experiments and Results

### 6.3.1 Experimental Setup

We evaluate our model for text-to-script mapping based on InScript and DeScript. We process the InScript and DeScript data sets using the Stanford Parser (Klein and Manning, 2003)[8]. We further resolve pronouns in InScript using annotated coreference chains from the gold standard. We individually test the two components, i.e. the identification of script-relevant verbs and event classification. Experiments on the first task are described in Section 6.3.2. Sections 6.3.3 and 6.3.4 present results on the latter task and a combination of both tasks, respectively. We split the InScript data into 810 texts for testing and 100 texts for tuning the CRF model threshold. All presented results are computed on the test part only.

For all models, we use pretrained 300-dimensional embeddings that are trained on the Google News corpus.[9]

[7] We also tried smoothing methods to counteract or at least weaken the effect, but this was not successful.

[8] To improve performance on the simplistic sentences from DeScript, we follow Regneri (2013) and re-train the parser

[9] Because our CRF model only supports nominal features, we discretize embeddings from code.google.com/archive/p/word2vec/ by binning the component values into three intervals $[-\infty, -\epsilon], [-\epsilon, \epsilon], [\epsilon, \infty]$. The hyperparameter $\epsilon$ is determined on a held-out development set.

| | P | R | $F_1$ |
|---|---|---|---|
| *Lemma* | 0.365 | **0.949** | 0.526 |
| *Decision Tree* | **0.628** | 0.817 | **0.709** |
| *Decision Tree (scenario independent)* | 0.513 | 0.877 | 0.645 |

**Table 6.1:** Identification of script-relevant verbs within a scenario and independent of the scenario. The maximum per column is printed in **bold**.

### 6.3.2 Event Verb Identification

In this evaluation, we test the ability of our model to identify verbs in narrative texts that instantiate script events. Our experiments make use of a 10-fold cross-validation setting within all texts of one scenario. To test the model in a scenario independent setting, we perform additional experiments based on a cross-validation with the 10 scenarios as one fold each and exclude the script features. That is, we repeatedly train our model on 9 scenarios and evaluate on the remaining scenario, without using any information about the test scenario. We compare the decision tree model to a baseline (Lemma) that always assigns the event class if the verb lemma is mentioned in DeScript. We report precision, recall and $F_1$-score on event verbs, averaged over all scenarios.

Table 6.1 gives an overview of the results based on 10-fold cross-validation. Our scenario specific model is capable of identifying more than 81% of script-relevant verbs at a precision of about 63%. This is a notable improvement over the baseline, which identifies 94.9% of the event verbs, but at a precision of only 36.5%. The table also gives numbers for the scenario independent setting: Precision drops to around 51% if only training data from other scenarios is available. One of the main difficulties here lies in classifying different non-script event verb classes in a way that generalizes across scenarios: Verbs that do not denote script events at all are usually easy to identify, since they are often just auxiliaries. Verbs marked as RelNScrEv are hard to identify in contrast, because they often describe events that are very close to events of the scenario in question. Modi et al. (2016) also found that distinguishing specific types of non-script events from script events can be difficult even for humans.

### 6.3.3 Event Type Identification

| | P | R | $F_1$ |
|---|---|---|---|
| *Lemma* | 0.516 | 0.442 | 0.475 |
| *Word2Vec* | 0.538 | 0.480 | 0.507 |
| *CRF model* | **0.623** | **0.485** | **0.543** |
| *CRF, no seq.* | 0.608 | 0.475 | 0.531 |

**Table 6.2:** Event Type Classification performance, for the similarity models ($sim_{lemma}$, based on lemma similarity, and $sim_{w2v}$, based on word2vec); and for the CRF model, with and without sequential features. The maximum per column is printed in **bold**.

In this section, we describe experiments on the event type identification task based on the subset of event instances from InScript that are annotated as script-related. As training data, we use the ESDs and the event type annotations from the DeScript gold standard[10]. The evaluation task is to classify individual event mentions in InScript based on their verbal realization in the narrative text. We evaluate against the gold-standard annotations from InScript. Since event type annotations are used for evaluation purposes only, this task comes close to a realistic setup, in which script knowledge is available for specific scenarios but no training data in the form of event-type annotated narrative texts exists. We report precision, recall and F1-scores, macro-averaged over all script-event types and scenarios.

Results for all models are presented in Table 6.2. Our CRF model achieves a F1-score of $0.543$, a considerably higher performance in comparison to the baseline models. As can be seen from excluding the sequential feature, ordering information improves the result. The rather small difference is due to the fact that ordering information can also be misleading (cf. Section 6.4). We found, however, that including the sequential feature accounts for an improvement of up to 4% in F1 score.

Table 6.3 shows $F_1$ scores for each individual scenario, for the CRF model with and without the sequential feature. As can be seen, ordering information increases the $F_1$ up to 4%, in

[10] In DeScript, there are some rare cases of EDs that do not describe a script event, but that are labeled as non-script event. We exclude these from the training data.

| Scenario | $F_1$ (no ordering info) | $F_1$ (ordering info) | Improvement |
|---|---|---|---|
| HAIRCUT | 0.532 | 0.521 | -0.011 |
| BICYCLE | 0.540 | 0.554 | 0.014 |
| BATH | 0.658 | 0.694 | 0.037 |
| BUS | 0.445 | 0.462 | 0.017 |
| TREE | 0.504 | 0.514 | 0.011 |
| TRAIN | 0.555 | 0.563 | 0.009 |
| FLIGHT | 0.476 | 0.478 | 0.003 |
| CAKE | 0.617 | 0.623 | 0.005 |
| GROCERY | 0.527 | 0.555 | 0.027 |
| LIBRARY | 0.460 | 0.465 | 0.005 |
| Average | 0.531 | 0.543 | 0.012 |

**Table 6.3:** The influence of the sequential feature per scenario.

the TAKING A BATH scenario. Only for GETTING A HAIRCUT, ordering information decreases performance. In this scenario, plausible event sequences are not covered by the script data (cf. Section 6.4). In general, ordering information is beneficial.

### 6.3.4 Full Script Parsing Task

| | P | R | $F_1$ |
|---|---|---|---|
| *Ident. model+Lemma* | 0.388 | 0.475 | 0.426 |
| *Ident. model+Word2vec* | 0.393 | **0.511** | 0.442 |
| *Ident. model+CRF model* | **0.458** | 0.505 | **0.478** |

**Table 6.4:** Full text-to-script mapping results. The maximum per column is printed in **bold**.

We now address the full text-to-script mapping task, a combination of the identification of relevant verbs and event type classification. This setup allows us to assess whether the general task of a fully automatic mapping of verbs in narrative texts to script events is feasible.

We compare the similarity models and the CRF model, but use them on top of our model for identifying script-relevant verbs instead of using the gold standard for identification. On the full text-to-script mapping task, our combined identification and CRF model achieves a precision and recall of $0.458$ and $0.505$, resp. (cf. Table 6.4). This reflects an absolute improvement over the baselines of $0.036$ and $0.052$ in terms of $F_1$-score. The results reflect the general difficulty of this task but are promising overall. As found in Chapter 4, even human annotators only achieve an agreement of 0.64 in terms of Fleiss' Kappa (Fleiss, 1971).

## 6.4 Discussion

### 6.4.1 Error Analysis

In this Section, we look in more detail at typical errors. We identified three major error sources.

- **Lexical Coverage.** We found that although DeScript is a small resource, training a model purely on ESDs works reasonably well. Coverage problems can be seen in cases of events for which only few EDs exist. An example is the CHOOSE_TREE event (the event of picking a tree at the shop) in the PLANTING A TREE scenario. There are only 3 EDs describing the event, each of which uses the event verb "choose". In contrast, we find that "choose" is used in less than 10% of the event mentions in InScript. Because of this mismatch, which can be attributed to the small training data size, more frequently used verbs for this event in InScript, such as "pick" and "decide", are labeled incorrectly. We observe that our meaning representation might be insufficient for finding synonyms for about 30% of observed verb tokens. This specifically includes scenario-specific and uncommon verbs, such as "squirt" in the context of the BAKING A CAKE scenario (squirt the frosting onto the cake). Problems may also arise from the fact that about 23% of the verb types occur in multiple paraphrase clusters of a scenario.

- **Misleading Ordering Information.** We found that ordering information is in general beneficial for text-to-script alignment. We however also identified cases for which it

can be misleading, by comparing the output of our full model to the model that does not use sequential features. As another result of the small size of DeScript, there are plausible event sequences that appear rarely or never at all in the training data. For the misclassifications due to misleading ordering information, this error accounts for approx. 60–70% of cases. An example is the WASH event in the GETTING A HAIRCUT scenario: It never appears directly after the MOVE_IN_SALON event (i.e. walking from the counter to the chair) in DeScript, but it is a plausible sequence that is misclassified by our model. In almost 15% of the observed errors, an event type is mentioned more than once, leading to misclassifications whenever ordering information is used. One reason for this might be that events in InScript are described in a more exhaustive or fine-grained way. For example, the WASH event in the TAKING A BATH scenario is often broken up into three mentions: wetting the hair, applying shampoo, and washing it again. However, because there is only one event type for the three mentions, this sequence is never observed in DeScript. Events with an interchangeable natural order lead to errors in a number of cases: In the BAKING A CAKE scenario, a few misclassifications occur because the order in which e.g. ingredients are prepared, the pan is greased and the oven is preheated is very flexible, but the model overfits to what it observed from the training. As last, there are also a few cases in which an event is mentioned, even before it actually takes place. In the case of the BORROWING A BOOK scenario, there are cases in InScript that mention in the first sentence that the purpose of the visit is to return a book. In DeScript in contrast, the RETURN event always takes place at the very end.

- **Near Misses.** For many verbs, it is also difficult for humans to come up with one correct event label. By inspecting confusion matrices for single scenarios, we found that for at least 3–5% of script event verbs in the test set, our model predicted an "incorrect" label for such verbs, but that label might still be plausible. In the BAKING A CAKE scenario, for example, there is little if any difference between mentions of making the dough and preparing ingredients. As a consequence, these two events are often confused: Approximately 50% of the instances labeled as PREPARE_INGREDIENTS are actually instances of MAKE_DOUGH.

### 6.4.2 Entailment Type and System Performance

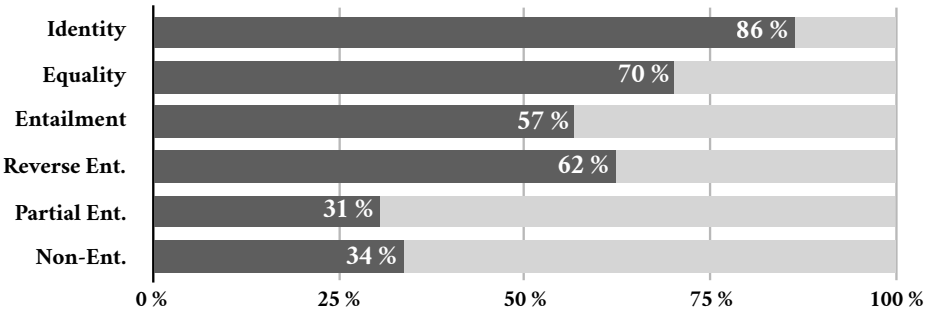



**Figure 6.4:** Accuracy of the CRF model on clausal entailment classes.

As additional analysis, we look at the accuracy of our event type identification model on the part of the data with annotated entailment classes from Chapter 5[11]: The results on all annotated clauses are shown in Figure 6.4.

*Identity* (verb and all head nouns are lemma-identical) is easiest to model, and so the model performs best on these cases. That the accuracy is not 100% is due to the fact that sometimes the same verb lemma occurs in different paraphrase sets. This holds in particular for light verbs such as *get*, which can be used to instantiate many different events throughout a scenario. In the BAKING A CAKE scenario, for example, *get* appears in the CHOOSE_RECIPE paraphrase set (*get your recipe*), as well as in GET_INGREDIENTS (*get a box of cake mix*), GET_UTENSILS (*get a pan*), etc.

In general, the important result is not the accuracy on the identity cases in itself, but rather that the high number of realization variants for event mentions contained in the paraphrase sets (38% Identity cases) help a lot with the mapping task.

To illustrate the positive effect of a large paraphrase set on system performance, we compare the result on Identity cases with a situation in which there is only one ED per paraphrase set. To this end, we picked one ED randomly from the paraphrase sets (average size is 25, using a token-based count), as representative(s) of the event type, and computed the number of verb identity cases automatically: They would drop from 58% to 26.9%. With a similar drop for participant identity cases, the resulting clause-based percentage of identity cases would

[11] The numbers don't exactly correspond to Ostermann et al. (2017), since there, the CRF model did not predict SCREV_OTHER because it does not appear in the entailment data.

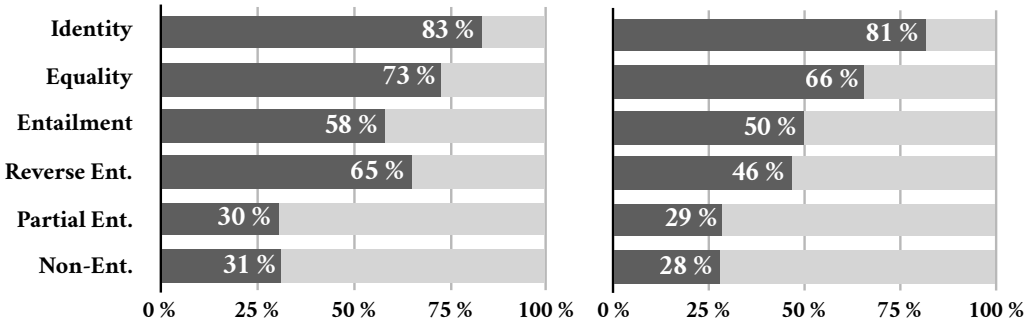



**Figure 6.5:** Accuracy of the baseline models on the entailment classes: $\text{sim}_{w2v}$ (left) and $\text{sim}_{lemma}$ (right).

be below 20% (instead of 38%), probably leading to a dramatic drop in labeling accuracy.

The cases we identified as complex are indeed most challenging for the model: Events in the *partial entailment* class are most difficult to map. This result provides a complementary message about the script knowledge base. About 80% of the clause-level *Partial Entailment* cases contain a word-level *Inference* relation, which is a strong indicator of the need for methods that are able to handle more complex inference: Crowdsourced collections of linguistic realization variants help to avoid complex inference in many cases, but cannot completely replace it. This emphasizes the need for larger script data collections that cover even more description variants for events and participants. The low performance on the *Equality* cases is mainly due to the fact that it subsumes the difficult *Diathesis* and *Phrasal Verb* relations. In contrast, the results for *Reverse Entailment* are better than one would expect: This is mostly due to the fact that most *Reverse Entailment* cases (approx. 85%) consist of combinations of *Equality* or *Identity* with an *Unrealized* participant. Even if one participant is unrealized, the correct label is mostly identified. For the text clause *I placed the tree, hole* is a required participant, occurring in every ED, but nevertheless the overlap is high enough to select the correct label. The more difficult *Hypernymy* cases make up only 15% of the *Reverse Entailment* cases. The situation is similar for the *Entailment* cases. Here, *Additional* participant cases make up 50%.

Finally, the model has low accuracy for the *Non-Entailment* cases. However, it is substantially above the random baseline. A possible reason is that the compositional computation of the clause-level entailment type amounts to a generalization to the worst case. Thus a number of pairs end up in the *Non-Entailment* class although there is only a minor local incompatibility.

As comparison, Figure 6.5 shows the accuracy of the baseline models on the annotated event verbs, which show a similar overall picture. The fact that the CRF outperforms $\text{sim}_{w2v}$ on *Equality* cases might be due to the fact that ordering information is beneficial in sorting out false positives: If two event types are equally plausible because two EDs are very similar, ordering information might help to pick the correct label. In contrast, ordering information can also be misleading in some cases, as we found before: $\text{Sim}_{w2v}$ even outperforms the CRF on *Equality*, *Entailment* and *Reverse Entailment* in terms of accuracy.

$\text{Sim}_{lemma}$ performs almost at par with the $\text{sim}_{w2v}$ on the *Equality* cases. An explanation for this is that these cases just require a lemma equality lookup.

## 6.5 Summary

In this chapter, we addressed the task of automatically mapping event denoting expressions in narrative texts to script events, based on an explicit script representation that is learned from crowdsourced data.

We present the first supervised script parsing model that is trained on crowdsourced data and thus applicable on a larger scale. We show that a model of script knowledge can be successfully trained on crowdsourced data, even if the number of training examples is small.

We find that our similarity-based baseline models already perform well given the high number of event labels, although they are not trained in a supervised fashion. Although the similarity models provide strong baselines, using a CRF, we are able to outperform them. We find that ordering information is especially helpful and that it improves the parsing performance.

We also closely inspected the performance of our parsing models, using the annotations of Chapter 5 as a diagnostic tool. Our main finding is that the size of the script database is crucial for the performance of a script parser: The larger the database is, the more formulation variants it provides - which improves the performance of a script parser trained on the script data.

Script parsing is a basic prerequisite for leveraging script knowledge in an actual natural language understanding application. Until now, script knowledge has only been evaluated in technical evaluation settings. In the next part of this dissertation, we present the first end-

to-end evaluation task for script knowledge, based on a machine comprehension corpus that requires systems to model script knowledge.

The findings of this chapter provide an important basis for leveraging script knowledge in NLU applications, and they provide an important outlook on the challenges that need to be taken to make script knowledge available for NLU models.

# Part III

# Script Knowledge for Machine Comprehension

Chapter 7

# Background and Motivation

In this part of the thesis, we will look at a new evaluation setting for script knowledge: Machine comprehension. In the past, script knowledge has mostly been tested in technical evaluations, i.e. by evaluating how complete or elaborate script representations are in comparison to a gold standard. While these evaluation efforts disclose the quality of the underlying script structure, they lack a connection to downstream tasks and actual applications.

We motivate the task of machine comprehension as an evaluation environment for script knowledge and commonsense knowledge. In the first part (Section 7.1), we take a closer look at story understanding, a special subtask of machine comprehension, in which a system has to process a narrative text and has to answer questions on it. Story understanding is closely tied with script knowledge inference, since a system needs to have certain commonsense inference capabilities about the narrative scenario to truly understand a text. We present both early and more recent story understanding models and corpora. While early systems explicitly focused on scripts, newer approaches rather concentrated on more general commonsense knowledge, such as knowledge about physical facts.

In Section 7.3, we look at evaluation methods that are focused on assessing how good models can draw commonsense inferences. In contrast to story understanding, the evaluation methods presented here do not correspond to actual applications and are less natural, but more focused on commonsense inference.

In the last part of Chapter 7 we look at related work in the area of machine comprehension in general. We look at standard machine comprehension corpora that focus on evaluating a systems' capability to process text, irrespective of its capability to perform commonsense reasoning. We also present standard machine reading models for the processing of such texts.

## 7.1 Machine Comprehension and Story Understanding

In this section, we lay some of the theoretical and methodological basics for understanding the idea of machine comprehension evaluations. In particular, we try to situate machine comprehension into the larger research landscape.[1]

An overarching goal of research in computational linguistics is to make a computer "understand" natural texts, a field that is referred to as Natural Language Understanding (NLU). One central challenge is to determine, given a text understanding model, whether the computer really "understands" a text or not. A good way to test human text understanding is to ask questions on a reading text, i.e. a *reading comprehension* task. The idea is adapted for computers: If a computer is able to correctly answer questions on a reading text, it is assumed that it understands the text. This task is also referred to as *machine comprehension*. Machine comprehension can be seen as a special kind of question answering on a reading text, as opposed to open question answering from large text collections, which is more connected to information retrieval.

There are several "flavours" of machine comprehension. For example, there are machine comprehension datasets on newswire texts, which require a system to model complex word meaning, or to look into different parts of a text at once in order to find an answer (Khashabi et al., 2018).

A different, particularly interesting case is *story understanding*, where the reading texts are narrative stories. Example 13 shows a snippet of a narrative text from the InScript corpus (s. Chapter 4) for illustration, together with two reading comprehension questions. A story understanding system that has processed the text should be able to find the correct answer

[1] Unfortunately, the terms of *machine comprehension*, *questions answering* and *natural language understanding* are often used interchangeably. The disposition presented here is thus subjective to some extent.

to both questions in order to show that it has truly understood the text.

(13) *Text:* I like to ride on trains because you can see lots of countryside without having to worry about driving. One time, when I lived in Alaska, I rode on a train from Fairbanks to Denali park. Just like with all train rides, I had to buy a ticket first and then I had to stand in line to wait for the train to stop at the station. Then I climbed onto the train and picked a seat. After the train started moving, a train attendant walked through to check our tickets and then I stood up to walk around. I saw lots of interesting things including a moose and we arrived at Denali right after that.
*Question 1:* Who checked his ticket?
*a.* a ticket inspector
*b.* the conductor
*Question 2:* Where did the person buy a ticket?
*a.* at the bord restaurant
*b.* at a ticket machine

Questions 1 and 2 illustrate two major challenges that need to be tackled for automated story understanding. A first challenge is the representation of words and their meaning **in the text**: To find an answer to the first question, a system needs to model word meaning, in order to find that *a ticket inspector* is a paraphrase of *a train attendant*, which is mentioned in the text. Similarly, the ability to resolve pronouns is beneficial, if the system can determine that *his* in Question 1 refers to the person riding the train. As mentioned above, modeling lexical meaning is a central aspect in most machine comprehension systems: This challenge is not specific to story understanding, but also to the wider field. Since understanding the meaning of words within a text usually does not require a system to perform reasoning over facts that are not mentioned in the text, we will henceforth refer to the inferences that a system needs to draw in order to answer such questions as **text-based inferences**.

Question 2 illustrates another challenge, namely the inference over facts that are not mentioned in the text, i.e. that go *beyond the text*. While a person is easily able to tell that *at a ticket machine* is the correct answer, this is challenging for a computer system, since neither *ticket machine* nor *board restaurant* are mentioned in the text. Information that goes beyond the text is needed in order to find the answer, in this case information that tickets are usually

bought at ticket machines rather than in bord restaurants. For narrative texts about everyday situations, these missing pieces of information can be events that are not mentioned but that are by default assumed to have happened; likewise participants of the activity that are not explicitly mentioned in the text, but that usually play a role in the scenario. In opposition to text-based inferences, we will henceforth refer to these kinds of inferences as **commonsense-based inferences**.

The commonsense knowledge that is needed to deeply understand such narratives is usually knowledge about everyday scenarios. This type of commonsense knowledge about everyday activities is script knowledge. Story understanding is thus an ideal evaluation ground for script knowledge. A text understanding system that has access to script knowledge should be able to answer question 2, while a system that has no access to such knowledge will struggle with the question. Unlike the script knowledge evaluation methods presented in Section 2, story understanding is a task-oriented kind of evaluation, i.e. it shows the usefulness of script knowledge in an actual application, namely machine comprehension.

### 7.1.1 Early Story Understanding Systems

In story understanding and general machine comprehension research, the focus in recent years has mostly been on text-based inferences, i.e. on finding ways to leverage lexical meaning, rather than on drawing inferences that go beyond the text content, which is a much harder problem. Most machine comprehension data sets that have been released in recent years (e.g. SQuAD, RACE, CNN/DailyMail, BAbI, CBT etc.) are large, i.e. they contain many questions in total, but only a small fraction of the questions requires commonsense-based inferences. (For a more elaborate overview, we refer to Section 7.2.)

Early work on story understanding tried to handle the problem of modeling commonsense-based inferences, but used only a small number of prepared demo texts. Most of the early approaches were based on a underlying, rule-based and scenario-specific script representation that was used to enhance text understanding[2].

*SAM* (Script Applier *M*echanism, Cullingford (1978)) is an example for such a system. SAM is a pattern-based script parser that uses script knowledge to understand stories in a re-

[2]The following paragraphs are based on Mueller (2006), a comprehensive review of older story understanding systems.

stricted number of domains. It is able to recognize when a script is activated by an event mention or a script-denoting expression. Over the text, it identifies participants and keeps track of their current states. It also recognizes events and which participants take part in events, building a script-augmented representation of the text. To answer questions about events and participants, it identifies a common event in text and question and uses information that is associated with this event. SAM works on a range of invented restaurant stories and some pre-edited newspaper texts.

Another example for an early system is the *BORIS* program (Dyer, 1982), which builds an "episodic memory representation of the story" containing facets of the story (e.g. scenario information, thematic information etc.) that are connected to hard-coded world knowledge. Like SAM, BORIS works only on a restricted set of texts of very restricted domains.

Miikkulainen (1993) proposed *DISCERN*, a neural network that can read and answer questions about script-based stories. The program uses memory models to store different aspects of script knowledge (causal event chains and role bindings of participants), and works on artificially created texts for 3 script scenarios (restaurant, shopping and travel).

Mueller (2004) presents an extension to the SAM program. Based on SAM, event calculus methods are used to learn a model of the text that can be used to answer questions. They especially focus on inferences involving time and space in a restricted number of domains. Unlike earlier approaches, their system does not rely on hand-written demo texts, but is the first script model that is applicable to arbitrary texts of the given scenario.

The earlier approaches on commonsense-based story understanding have in common that they only work on manually constructed test cases and were made for demonstration purposes rather than for actual applications. The used scenarios are predefined, and solely inferences on these narrow domains are modeled. Moreover, most of the systems encode script knowledge in a rule based or pattern based fashion, which means that they do not work well on new texts. This restriction is critical particularly in comparison to newer machine comprehension systems for text-based inference: Such systems work on large text collections and many domains, while early approaches are not applicable to real-world tasks.

### 7.1.2 Machine Comprehension Datasets with a Focus on Commonsense Reasoning

Only in the last 2-3 years, efforts have been made to provide evaluation data for commonsense-based machine comprehension on a wider range of scenarios. While the corpora described below contain more natural texts than early evaluation data, they are not story understanding evaluation methods in the narrow sense: They don't focus on the impact of script knowledge reasoning and their genre is usually not narrative texts. They still require a system to infer information that is not mentioned in the reading text explicitly, but usually assume a broader notion of commonsense knowledge, as it is for example encoded in commonsense knowledge databases such as WebChild (Tandon et al., 2017) or ConceptNet (Speer et al., 2017) (cf. Chapter 2). Note that the majority of the introduced data sets was created after the publication of our MCScript data, which is described in Chapter 8.

**News Domain.** Two recently published machine comprehension data sets that require commonsense-based inference are based on news texts. First, *NewsQA* (Trischler et al., 2017) is a dataset of newswire texts from CNN with questions and answers written by crowd workers. During data collection, full texts were not shown to workers as a basis for question formulation, but only the text's title and a short summary, to avoid literal repetitions and support the generation of non-trivial questions requiring background knowledge. The task is the identification of document passages containing the answer. Knowledge required to answer the questions is mostly factual knowledge.

Figure 7.1 gives an example text from NewsQA with a question. Both green text spans are possible answers to the question. To find the answers, a system needs to infer that *challenging a law* can result in *slamming it*.

Second, *ReCoRD* (Zhang et al., 2018) is a gap-filling task on newswire texts. The gap-filling tests were not crowdsourced, but automatically created, by hiding a named entity in a larger passage from the text. The passage serves as "question", and the hidden entity needs to be predicted. The authors show that commonsense inference is needed for a major part of the questions, since the deleted paragraphs are usually quite long, such that machine comprehension systems will have to make many successive inference steps.

| |
|---|
| *Text:* (CNN) – Four groups that advocate for immigrant rights said Thursday they will challenge Arizona's new immigration law, which allows police to ask anyone for proof of legal U.S. residency. The Mexican American Legal Defense and Educational Fund, the American Civil Liberties Union, the ACLU of Arizona and the National Immigration Law Center held a news conference Thursday in Phoenix to announce the legal challenge. … |
| *Question:* Who slammed the law? |

**Figure 7.1:** Example text fragment from NewsQA

| |
|---|
| *Text:* … Back then, Salameh had no expedition experience, little money and couldn't have pointed to Mount everest on a map. Fast forward to 2017, and he has become the first Jordanian to complete the "Explorers Grand Slam", which entails climbing the Seven Summits, including Mount Everest, and reaching the North and South poles. … |
| *Question:* Almost immediately afterward, Salameh stopped partying, smoking and began training for the X. |

**Figure 7.2:** Example text fragment from ReCoRD

Figure 7.2 gives an example text fragment from ReCoRD with a question. To find the correct answer, a system has to infer long and complicated chain of commonsense facts: The *Explorers Grand Slam* will require lots of training, which will require a healthy lifestyle, without partying and smoking.

**Web Texts.** Other corpora don't use texts in the news domain, but web documents. They include for example *TriviaQA* (Joshi et al., 2017), a corpus that contains automatically collected question-answer pairs from 14 trivia and quiz-league websites, together with web-crawled evidence documents from *Wikipedia* and *Bing*. While a majority of questions require world knowledge for finding the correct answer, it is mostly factual knowledge. In addition, the genre of the evidence documents is unrestricted, i.e. different types of texts are used. Figure 7.3 gives an example question from TriviaQA along with an evidence document snippet

| |
|---|
| *Text (from Wikipedia):* … Eliza is a Cockney flower girl, who comes to Professor Henry Higgins asking for elocution lessons, after a chance encounter at Covent Garden. Higgins goes along with it for the purposes of a wager: That he can turn her into the toast of elite London society. … |
| *Question:* Who taught Eliza Dolittle to be a lady? - Professor Henry Higgins. |

**Figure 7.3:** Example text fragment from TriviaQA

from Wikipedia. The challenges here are two-fold: (1) A system needs to draw the complicated commonsense inference that Prof. Higgins gives elocution lessons and turns Eliza "into the toast of elite society", means that he teaches her to be a lady. (2) The system needs to find this relevant text part from a large text, namely a whole Wikipedia article or a web document, which has in some respects more resemblance to an information retrieval task than machine comprehension.

*CommonsenseQA* (Talmor et al., 2019) contains a total of over 9000 multiple-choice questions that were crowdsourced based on triples of $(entity_q, relation, entity_a)$ in ConceptNet: Workers had to formulate a question about $entity_q$ that could be answered with $entity_a$, requiring a system to model the $relation$ between the entities. To generate wrong answers, triples $(entity_q, relation, entity_a^2)$ were extracted from ConceptNet, and $entity_a^2$ was assumed to be an incorrect answer that is semantically similar to $entity_a$ due to the same relation to $entity_q$. Texts were only added post-hoc, by querying a web search engine based on the formulated question, and by adding the retrieved evidence texts to the questions and answers.

**Fiction.** *NarrativeQA* is a reading comprehension dataset that largely differs from other corpora in terms of text length. Instead of providing short reading texts, systems have to process complete novels or movie scripts and answer very complex questions. About 30 questions per novel were crowdsourced based on book summaries, together with matching answers. Two tasks are addressed: (1) predicting answers and (2) finding the correct answer for each question from the 30 possible answers per novel. Answering questions in

NarrativeQA doesn't necessarily require world knowledge, but systems need to be able to spot relevant passages in large, coherent collections of text. The task is similar to an information retrieval task: Once the relevant passages or chapters in the novel are identified, the information needed to find the answer can usually be extracted from them.

Story understanding is an ideal end-to-end evaluation ground for script knowledge. Earlier approaches provided only demonstrator cases and no proper evaluation methods. More recent tasks and corpora handle different domains, which results in the fact that they evaluate a much broader notion of commonsense inference, including e.g. physical knowledge (for trivia texts) and knowledge about politics (for newswire texts).

What is missing, is a story understanding task based on everyday texts, which goes beyond early demonstrator tasks and allows for a reliable assessment of the impact of script knowledge. To provide such an evaluation, we created the *MCScript* and *MCScript2.0* corpora, described in Chapter 8 and Chapter 10. Unlike previous works, our corpora provide a range of *narrative* texts from a large number of everyday scenarios together with multiple-choice questions, a large proportion of which require reasoning over everyday commonsense knowledge. Our resources are thus the first actual evaluation data sets for this kind of knowledge in the area of story understanding.

In the following sections, we will give short overviews of two areas that are related to story understanding: First, we give a short overview of data sets and systems for purely text-based machine comprehension, which do not explicitly require systems to perform commonsense-based reasoning. Second, we look at other evaluation paradigms for commonsense inference, i.e. tasks that test a system's capability to draw commonsense inferences, but not in a reading comprehension setting.

## 7.2 Text-Based Machine Comprehension Tasks and Systems

In this section, we look at related work in the area of general machine comprehension, without a focus on commonsense inferences. We introduce the most common corpora and models for purely text-based inference, i.e. with no special focus on including external knowledge.

### 7.2.1 Data Sets

In recent years, a number of reading comprehension datasets for text-based understanding have been proposed, which differ with respect to text type (Wikipedia texts, examination texts, etc.), mode of answer selection (span-based, multiple choice, etc.) and test systems with regard to different aspects of language understanding, without explicitly addressing commonsense knowledge.

**Multiple-Choice Questions.** *MCTest* (Richardson et al., 2013) was one of the first machine comprehension data sets with a focus purely on text-based understanding. Based on crowd-sourced fictional stories, a larger portion of the questions requires methods for combining information from various sentences in order to find the correct answer to multiple choice questions.

In the same spirit, *MultiRC* contains multiple-choice questions that require a system to process several text sentences in order to identify one or more correct answer from a set of several multiple-choice answers. The dataset comprises of reading passages from 7 different genres.

*RACE* (Lai et al., 2017) is based on examination tests, and provides candidate answers to the multiple-choice questions. It is similar to MCTest in style, but larger in size.

**Gap Filling Tasks.** The *Children's Book Test* (*CBT*, Hill et al. (2015)) contains gap-filing questions on children's books. Systems are required to process larger parts of a text and to infer a named entity that is missing from a target sentence.

*CNN/Daily Mail* (Hermann et al., 2015) is a large corpus of news texts, that also contains cloze tests for named entities.

**Other Data Sets.** The *Stanford Question Answering Dataset* (*SQuAD*, Rajpurkar et al. (2016)) is a large machine comprehension dataset based on Wikipedia texts. The task is to predict a span in the text that answers the questions.

*BAbI* (Weston et al., 2015) is an artificial machine comprehension framework containing 20 different toy tasks that each test a single aspect of machine comprehension, such as the

ability to resolve pronouns, the ability to count, or the ability to extract single or multiple supporting facts from a text.

### 7.2.2 Systems

Since a large number of reading comprehension systems has been developed in the past few years, we will only very briefly highlight some of the more prominent systems here.

Many recent machine comprehension systems are based on recurrent neural networks (*RNNs*). The *Attentive Reader* is such a model, and a generalization of memory networks (Weston et al., 2014) to reading comprehension. It was one of the first reading comprehension models that utilizes an attention mechanism. As such, many recent machine comprehension models are built on its basic architecture, which is why we describe it here in greater detail[3]. The model consists of two recursive bi-directional long short-term memory readers (henceforth *LSTM*, (Hochreiter and Schmidhuber, 1997)) that process text and question, respectively. The text is represented as a weighted average of the hidden states of the text LSTM. The weights for each token are learned by combining the respective hidden state at the token position with the question representation, which is just the last hidden state of the question LSTM. In other words, the question is used to attend to the text tokens, and to decide which of the tokens is more important. Text and question representation are then combined for classification. In the original model, a named entity is predicted, but there exist other variants (e.g. Lai et al. (2017), who use the model to choose one of several answer candidates). Chen et al. (2016) present an extension of the model, using bilinear attention computations and GRUs instead of LSTMs.

The *Gated-Attention Reader* (Dhingra et al., 2017) follows the paradigm of using recursive neural network modules to process text and question. It employs several layers of attention that are built on top of each other (*multi-hop* architecture), i.e. the question does not attend to the text only once, but in each layer, based on the previous attended representation. Also, the attention computation is slightly modified. The multi-hop architecture is intended to encode deeper kinds of relations between the word than the shallow attention formulation of the Attentive Reader.

[3] For a formal description of the model, s. Chapter 9.

*Transformers* have been proposed as an alternative to RNN-based models, because the latter usually require a long training time due to their complexity. Vaswani et al. (2017) use feed-forward networks and self attention layers to replace RNN reader components in machine translation. Self attention is a simplification of the previously described attention formalisms: The idea is to use the text itself to compute attention weights. The weights thus no longer highlight words that are important for a question, but just generally prominent tokens[4]. Yu et al. (2018) extend this transformer model with convolution layers and adapt it to the reading comprehension task. Their model, called *QANet*, reached the state of the art on SQuAD at the time of publication.

## 7.3 Other NLU Tasks for Evaluating Commonsense Inference

In this section, we widen the focus again, moving away from machine comprehension and question answering as evaluation method for commonsense reasoning. There have been other tasks in natural language understanding that are used to evaluate the commonsense reasoning capabilities of a system, which we will briefly introduce here.

The *ROCStories* corpus (Mostafaza deh et al., 2016) is an example for an alternative story understanding evaluation. It contains 100,000 crowdsourced narrations about everyday scenarios. All stories consist of 5 sentences. Mostafaza deh et al. (2016) propose the *Story Cloze* test on their data set: The last sentence is left out, and a (new) plausible and an implausible ending are crowdsourced in an additional experiment. Systems have to decide between the two endings. The authors provide more than 3,500 gap-filling tasks for testing. For training models, the remaining full texts can be used, which means there are no negative examples for training. Due to its domain and setup, the story cloze test requires a system to use commonsense knowledge about everyday scenarios. However, the texts are topically unrestricted and very short (i.e. mentioning only one or two concrete events per text), which makes it difficult to evaluate the relevance of script knowledge specifically. It has been shown that simple models that just use stylistic features on the endings, ignoring

[4] Self attention can thus be seen as a sort of filter mechanism, that sorts out tokens that are not important, or stop words.

the preceding sentences, work well on the data.

| |
|---|
| *Text:* Karen was assigned a roommate her first year of college. Her roommate asked her to go to a nearby city for a concert. Karen agreed happily. The show was absolutely exhilarating. |
| *Right Ending:* Karen became good friends with her roommate.<br>*Wrong Ending:* Karen hated her roommate. |

**Figure 7.4:** Example text fragment from TriviaQA

Figure 7.4 shows an example item from the story cloze test. The story talks about a girl that attends college and visits a concert with a roommate. To find the correct ending, a system should have the reasoning capability to know that it is very unlikely that Karen does not like her roommate after visiting an "exhilarating" concert with her. A very general kind of commonsense knowledge is required: If two persons experience something exhilarating together, it is probable that they like each other.

*SWAG* (Zellers et al., 2018) is a natural language inference corpus requiring the use of commonsense knowledge. It contains 113,000 cloze-style multiple choice test items in the form of short video descriptions missing the last verb phrase. The task is to predict the missing verb phrase from a range of possibilities. The possible answers consist of the original verb phrase and 3 alternative verb phrases that are generated with a language model by massive oversampling and then filtering the samples with a method called *adversarial filtering*. The video descriptions that are used to generate the test instances were taken from video corpora about different everyday activities, thus the domain requires systems to know about everyday commonsense knowledge. Similar to *CommonsenseQA*, *SWAG* however does not contain reading texts, and no textual material is used apart form the cloze-style question.

Another evaluation methodology for everyday commonsense knowledge is the *Winograd Schema Challenge* (Levesque et al., 2012). A Winograd schema always consist of a sentence with several discourse referents and a single binary question that has one of the referents as an answer. The kind of commonsense knowledge that is required to answer the question is usually specific to the nouns in question, and is only needed to support referent resolution to find the answer. No full text is used to support the inference.

| |
|---|
| *Text:* The trophy would not fit in the brown suitcase because it was too big. What was too big? |
| *Right Answer:* the trophy<br>*Wrong Ending:* the suitcase |

**Figure 7.5:** Example item from the Winograd schema challenge

Figure 7.5 shows an example item from the Winograd schema challenge. The task is to find which discourse referent is too big, which requires physical knowledge.

The *COPA* corpus contains 1000 handcrafted short premises, followed by two alternative continuations. The premises and endings are based on personal stories from the web and require a system to have causal world knowledge, e.g. the fact that *losing balance on a ladder* causes *falling off the ladder*.

## 7.4 Summary

In this chapter, we motivated the task of machine comprehension for the evaluation of script knowledge.

We first looked at the task of story understanding. While early work on story understanding concentrated on evaluating script knowledge, the data and systems developed were mere demonstrators and not actually applicable in an end-to-end application. Recent work on story understanding has focused on other genres than narratives, such as news texts and web texts. The kind of commonsense knowledge that is investigated here is thus more general and reaches from knowledge about politics to physical knowledge up to facts about the world.

Second, we presented general text-based machine comprehension data sets and models.

Third, we looked at alternative evaluation tasks for commonsense knowledge reasoning. While there exist several approaches to evaluate commonsense inference in a textual entailment or coreferent resolution task, none of them explicitly addresses script knowledge.

To sum up, we find that recent work on machine comprehension concentrates on a very general type of commonsense knowledge inferences, or often is not focused on such inferences

at all. This leaves open the important question to what extent script knowledge is helpful for text understanding. In contrast, early work on the topic was more focused on scripts, but provided only a small number of demonstrator cases.

In the next chapters, we attempt to close this gap by providing data and models for the focused evaluation of the contribution of script knowledge for natural language understanding. We present MCScript and MCScript2.0, the first datasets focused on evaluating the contribution of script knowledge for natural language understanding in an actual application.

Chapter 8

# MCScript: A Dataset for Assessing the Contribution of Script Knowledge for Machine Comprehension

In Chapter 2, we introduced the most common evaluation tasks for script knowledge representations. What these evaluations all have in common is that they intrinsically evaluate the script structure, without proving its usefulness in an application. As found in Chapter 7, story understanding would be such an application, providing an ideal evaluation for models of script knowledge. However, there exists no corpus for the end-to-end evaluation of the contribution of script knowledge for story understanding models. In this chapter, we close this gap and describe a research effort aimed at assessing the usefulness of script knowledge for a natural language understanding application, namely machine comprehension. We introduce *MCScript*, a dataset to evaluate natural language understanding approaches with a focus on interpretation processes requiring inference based on script knowledge. Our framework makes it possible to assess system performance in a multiple-choice question answering setting, without imposing any specific structural or methodical requirements.

MCScript is a collection of (1) narrative texts, (2) questions of various types referring to these texts, and (3) pairs of answer candidates for each question. It comprises approx. 2,100 texts

| | |
|---|---|
| **T** | I wanted to plant a tree. I went to the home and garden store and picked a nice oak. Afterwards, I planted it in my garden. |
| **Q1** | What was used to dig the hole? |
| | a. a shovel b. his bare hands |
| **Q2** | When did he plant the tree? |
| | a. after watering it b. after taking it home |

**Figure 8.1:** An example for a text snippet with two reading comprehension questions.

and a total of approximately 14,000 questions. Answering a substantial subset of questions requires knowledge beyond the facts mentioned in the text, i.e. it requires inference using commonsense knowledge about everyday activities. An example is given in Figure 8.1. For both questions, the correct choice for an answer requires knowledge about the activity of planting a tree, which goes beyond what is mentioned in the text. Texts, questions, and answers were obtained through crowdsourcing. In order to ensure high quality, we validated and filtered the dataset. Due to our design of the data acquisition process, we ended up with a substantial subset of questions that require commonsense inference (27.4%).

The corpus was used as a basis for a shared task on machine comprehension using commonsense knowledge, that was co-organized at SemEval 2018 by the author of this dissertation. Participants of the task were explicitly encouraged to use commonsense knowledge resources in addition to the training data. We provided script data collections (DeScript and OMCS), but also explicitly pointed to other script representations and data sets, such as narrative chains or event embeddings (Modi and Titov, 2014). The systems that participated in the shared task are presented and discussed in Chapter 9.

The structure of this chapter is as follows:

- We first present a series of pilot studies that we conducted in order to collect commonsense inference questions (Section 8.1).
- In Section 8.2, we discuss the data collection of the three main components required for a machine comprehension data set: Questions, texts and answers. The data were collected via crowdsourcing on Amazon Mechanical Turk[1] (henceforth *MTurk*).

[1] www.mturk.com

- We provide a thorough analysis of MCScript and give various statistics in Section 8.3.

## 8.1 Pilot Study

As a starting point for our pilots, we made use of texts from InScript, which provides stories centered around everyday situations (see Section 8.2.2). We conducted three different pilot studies to determine the best way of collecting questions that require inference over commonsense knowledge:

The most intuitive way of collecting reading comprehension questions is to show texts to workers and let them formulate questions and answers on the texts, which is what we tried internally in a *first pilot*. Since our focus is to provide an evaluation framework for inference over commonsense knowledge, we manually assessed the number of questions that indeed require common sense knowledge. We found too many questions and answers collected in this manner to be lexically close to the text.

In a *second pilot*, we investigated the option to take the questions collected for one text and show them as questions for another text of the same scenario. While this method resulted in a larger number of questions that required inference, we found that the majority of questions made no sense at all when paired with another text. Many questions were specific to a text (and not to a scenario), requiring details that could not be answered from other texts.

Since the two previous pilot setups resulted in questions that centered around the texts themselves, we decided on a *third pilot* and not to show workers any specific texts at all. Instead, we asked for questions that centered around a specific script scenario (e.g. EATING IN A RESTAURANT). We found this mode of collection to result in questions that have the right level of specificity for our purposes: namely, questions that are related to a scenario and that can be answered from different texts (about that scenario), but for which a text does not need to provide the answer explicitly.

The next section will describe the mode of collection chosen for the final dataset, based on the third pilot, in more detail.

## 8.2 Data Collection

### 8.2.1 Scenario Selection

As mentioned in the previous section, we decided to base the question collection on script scenarios rather than specific texts. As a starting point for our data collection, we use scenarios from three script data collections (Regneri et al., 2010; Singh et al., 2002; Wanzare et al., 2016). Together, these resources contain more than 200 scenarios. To make sure that scenarios have different complexity and content, we selected 80 of them and came up with 20 new scenarios. Together with the 10 scenarios from InScript, we ended up with a total of 110 scenarios.

### 8.2.2 Texts

For the collection of texts, we followed the collection process of InScript, where workers were asked to write a story about a given activity "as if explaining it to a child". This results in elaborate and explicit texts that are centered around a single scenario. Consequently, the texts are syntactically simple, facilitating machine comprehension models to focus on semantic challenges and inference. We collected 20 texts for each scenario. Each participant was allowed to write only one story per scenario, but could work on as many scenarios as they liked. For each of the 10 scenarios from InScript, we randomly selected 20 existing texts from that resource.

### 8.2.3 Questions

For collecting questions, workers were instructed to "imagine they told a story about a certain scenario to a child and want to test if the child understood everything correctly". This instruction ensured that questions are linguistically simple, elaborate and explicit. Workers were asked to formulate questions about details of such a situation, i.e. independent of a concrete narrative. This resulted in questions, where the answer is not literally mentioned in the text.

To cover a broad range of question types, we asked participants to write 3 temporal questions (asking about time points and event order), 3 content questions (asking about persons or details in the scenario) and 3 reasoning questions (asking how or why something happened). They were also asked to formulate 6 free questions, which resulted in a total of 15 questions. Asking each worker for a high number of questions enforced that more creative questions were formulated, which go beyond obvious questions for a scenario.

Since participants were not shown a concrete story, we asked them to use the neutral pronoun "they" to address the protagonist of the story. We permitted participants to work on as many scenarios as desired and we collected questions from 10 participants per scenario.

### 8.2.4 Answers

Our mode of question collection results in questions that are not associated with specific texts. For each text, we collected answers for 15 questions that were randomly selected from the same scenario. Since questions and texts were collected independently, answering a random question is not always possible for a given text. Therefore, we carried out answer collection in two steps. In the first step, we asked participants to assign a category to each text–question pair.

We distinguish two categories of answerable questions: The category *text-based* was assigned to questions that can be answered from the text directly. If the answer could only be inferred by using commonsense knowledge, the category *script-based* was assigned. Making this distinction is interesting for evaluation purposes, since it enables us to estimate the number of commonsense inference questions. For questions that did not make sense at all given a text, *unfitting* was assigned. If a question made sense for a text, but it was impossible to find an answer, the label *unknown* was used.

In a second step, we asked participants to formulate a plausible correct and a plausible incorrect answer candidate to answerable questions (*text-based* or *script-based*). To level out the effort between answerable and non-answerable questions, participants had to write a new question when selecting *unknown* or *unfitting*.

In order to get reliable judgments about whether or not a question can be answered, we collected data from 5 participants for each question and decided on the final category via

majority vote (at least 3 out of 5). Consequently, for each question with a majority vote on either *text-based* or *script-based*, there are 3 to 5 correct and incorrect answer candidates, one from each participant who agreed on the category. Questions without a clear majority vote or with ties were not included in the dataset.

### 8.2.5 Data Post-Processing

We performed four post-processing steps on the collected data.

- We manually filtered out texts that were instructional rather than narrative.
- All texts, questions and answers were spellchecked by running *aSpell*[2] and manually inspecting all corrections proposed by the spellchecker.
- We found that some participants did not use "they" when referring to the protagonist. We identified "I", "you", "he", "she", "my", "your", "his", "her" and "the person" as most common alternatives and replaced each appearance manually with "they" or "their", if appropriate.
- We manually filtered out invalid questions, e.g. questions that are suggestive ("Should you ask an adult before using a knife?") or that ask for the personal opinion of the reader ("Do you think going to the museum was a good idea?").

### 8.2.6 Answer Selection and Validation

We finalized the dataset by selecting one correct and one incorrect answer for each question–text pair. To increase the proportion of non-trivial inference cases, we chose the candidate with the *lowest* lexical overlap with the text from the set of correct answer candidates as *correct* answer. Using this principle also for incorrect answers leads to problems. We found that many incorrect candidates were not plausible answers to a given question. Instead of selecting a candidate based on overlap, we hence decided to rely on majority vote and selected the candidate from the set of incorrect answers that was most often mentioned.

[2] http://aspell.net/

For this step, we normalized each candidate by lowercasing, deleting punctuation and stop words (articles, *and*, *to* and *or*), and transforming all number words into digits, using *text2num*[3]. We merged all answers that were string-identical, contained another answer, or had a Levenshtein distance (Levenshtein, 1966) of 3 or less to another answer. The "most frequent answer" was then selected based on how many other answers it was merged with. Only if there was no majority, we selected the candidate with the highest overlap with the text as a fallback.

Due to annotation mistakes, we found a small number of chosen correct and incorrect answers to be inappropriate, that is, some "correct" answers were actually incorrect and vice versa. Therefore, we manually validated the complete dataset in a final step. We asked annotators to read all texts, questions, and answers, and to mark for each question whether the correct and incorrect answers were appropriate. If an answer was inappropriate or contained any errors, they selected a different answer from the set of collected candidates. For approximately 11.5% of the questions, at least one answer was replaced. 135 questions (approx. 1%) were excluded from the dataset because no appropriate correct or incorrect answer could be found.

## 8.3 Data Statistics and Analysis

For all experiments, we admitted only experienced MTurk workers who are based in the US. One HIT[4] consisted of writing one text for the text collection, formulating 15 questions for the question collection, or finding 15 pairs of answers for the answer collection. We paid $0.50 per HIT for the text and question collection, and $0.60 per HIT for the answer collection.

More than 2,100 texts were paired with 15 questions each, resulting in a total number of approx. 32,000 annotated questions. For 13% of the questions, the workers did not agree on one of the 4 categories with a 3 out of 5 majority, so we did not include these questions in our dataset.

The distribution of category labels on the remaining 87% is shown in Table 8.1. 14,074

[3] https://github.com/ghewgill/text2num

[4] A *Human Intelligence Task* (HIT) is one single experimental item in MTurk.

| answerable | | not answerable | |
|---|---|---|---|
| text-based | script-based | unknown | unfitting |
| 10,160 | 3,914 | 9,974 | 3,172 |
| 14,074 | | 13,246 | |

**Table 8.1:** Distribution of question categories

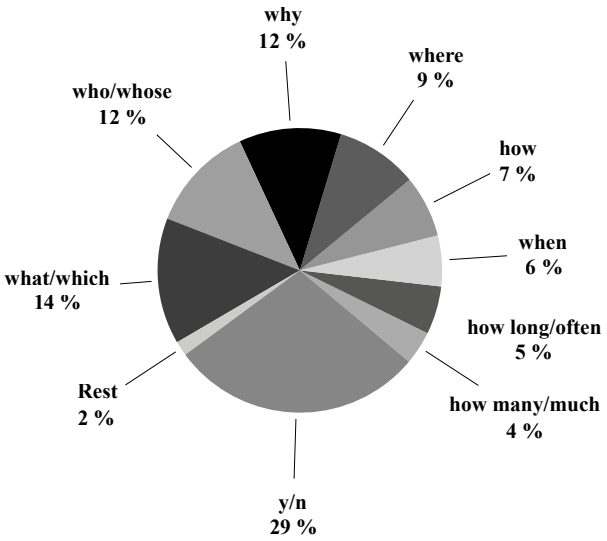



**Figure 8.2:** Distribution of question types in the data.

(52%) questions could be answered. Out of the answerable questions, 10,160 could be answered from the text directly (*text-based*) and 3,914 questions required the use of commonsense knowledge (*script-based*). After removing 135 questions during the validation, the final dataset comprises 13,939 questions, 3,827 of which require commonsense knowledge (i.e. 27.4%). This ratio was manually verified based on a random sample of questions.

We split the dataset into training (9,731 questions on 1,470 texts), development (1,411 questions on 219 texts), and test set (2,797 questions on 430 texts). Each text appears only in one of the three sets. The complete set of texts for 5 scenarios was held out for the test set.

The average text, question, and answer length is 196.0 words, 7.8 words, and 3.6 words, respectively. On average, there are 6.7 questions per text.

Figure 8.2 shows the distribution of question types in the dataset, which we identified using simple heuristics based on the first words of a question: Yes/no questions were identified as

**T** It was time to prepare for the picnic that we had plans for the last couple weeks. …I needed to set up the cooler, which included bottles of water, soda and juice to keep everyone hydrated. Then I needed to ensure that we had all the food we intended to bring or cook. So at home, I prepared baked beans, green beans and macaroni and cheese. …But in a cooler, I packed chicken, hotdogs, hamburgers and rots that were to be cooked on the grill once we were at the picnic location.

**Q1** What did they bring to drink?

a. Water, soda and juice.
b. Water, wine coolers and sports drinks.

**Q2** What type of food did they pack?

a. Meat, drinks and side dishes.
b. Pasta salad only.

**Figure 8.3:** An example text with 2 questions from MCScript

questions starting with an auxiliary or modal verb, all other question types were determined based on the question word.

We found that 29% of all questions are yes/no questions. Questions about details of a situation (such as *what/ which* and *who*) form the second most frequent question category. Temporal questions (*when* and *how long/often*) form approx. 11% of all questions. We leave a more detailed analysis of question types for future work.

As can be seen from the data statistics, our mode of collection leads to a substantial proportion of questions that require inference using commonsense knowledge. Still, the dataset contains a large number of questions in which the answer is explicitly contained or implied by the text: Figure 8.3 shows passages from an example text of the dataset together with two such questions. For question Q1, the answer is given literally in the text. Answering question Q2 is not as simple; it can be solved, however, via standard semantic relatedness information (chicken and hotdogs are meat; water, soda and juice are drinks).

The following cases require commonsense inference to be decided. In all these cases, the

answers are not overtly contained nor easily derivable from the respective texts. We do not show the full texts, but only the scenario names for each question.

(14) BORROWING A BOOK FROM THE LIBRARY

Did they have to pay anything to borrow the book?

a. yes

b. no

(15) CHANGING A BABY DIAPER

Did they throw away the old diaper?

a. Yes, they put it into the bin.

b. No, they kept it for a while.

(16) CLEANING THE TABLE

When did they clean the table?

a. After a meal

b. Before they ate

(17) PREPARING A PICNIC

Who is packing the picnic?

a. the children

b. the parents

(18) TAKING A SHOWER

How long did the shower take?

a. a few hours

b. a few minutes

Example 14 refers to a library setting. Script knowledge helps in assessing that usually, PAY is not an event when borrowing a book, which answers the question. Similarly, event information helps in answering the questions in Examples 15 and 16. In Example 17, knowledge about the typical role of parents in the preparation of a picnic will enable a plausibility decision. Similarly, in Example 18, it is commonsense knowledge that showers usually take a few minutes rather than hours.

(19) MAKING BREAKFAST

What time of the day is breakfast eaten?

a. at night

b. in the morning

There are also cases in which the answer can be inferred from the text, but where commonsense knowledge is still beneficial: The text for example 19 does not contain the information that breakfast is eaten in the morning, but it could still be inferred from many pointers in the text (e.g. phrases such as *I woke up*), or from commonsense knowledge.

These few examples illustrate that our dataset covers questions with a wide spectrum of difficulty, from rather simple questions that can be answered from the text to challenging inference problems.

## 8.4 Conclusion

In this chapter, we presented MCScript, the first corpus for an end-to-end evaluation of script knowledge. MCScript makes it possible to evaluate script knowledge representations in a naturalistic application, which is an advantage as compared to the intrinsic script knowledge evaluations conducted in prior work.

The texts that are used for the corpus are narrations about 110 everyday scenarios. Questions for the data set were collected based on script scenarios, rather than individual texts, which resulted in question - answer pairs that explicitly require reasoning over script knowledge. Previous evaluation tasks were more focused on text-based inferences, or on more general commonsense knowledge. Our data collection instead provides a setup that is intended to assess the contribution of script knowledge for computational models of language understanding in a real world evaluation scenario, for the first time.

In the next chapter, we look at the performance of various benchmark models on the data. We will also look at the performance of a range of models that were proposed in a shared task which used MCScript as underlying evaluation data.

Chapter 9

# Experiments on MCScript

In the last chapter, we presented MCScript, a dataset for evaluating script knowledge in a machine comprehension setting. MCScript is meant to test a system's capability to perform inference steps over script knowledge. Consequently, one would expect that a machine comprehension system that has access to script knowledge should perform better than a system that has no such access. One would also expect that questions which are annotated as script knowledge-based are more difficult to answer than questions that are annotated as text-based.

To test these hypotheses, we first present a range of standard baseline models that have no access to script knowledge, and discuss their performance, in order to assess the general difficulty of the task. Next, we report on the results of a SemEval shared task that was conducted using the MCScript corpus, attracting a total of more than 250 registrations and 11 submissions. We present the results of systems that were submitted for the task, which includes systems that made use of ConceptNet as a resource for commonsense knowledge, and discuss their performance.

Unfortunately, the central results of our evaluation do not verify our hypotheses: While our baselines struggle with the general difficulty of the data, we find that questions that are annotated as script-based are not necessarily harder to answer than text-based questions. Second, in contrast to our expectations, we find that the use of script knowledge is

not helpful for the systems that participated in the shared task. The effect of using a large commonsense knowledge graph like ConceptNet was found to be positive, but small. For humans, the task is nevertheless trivial.

We take a closer look at one of the participating systems that performed very well without using any additional commonsense knowledge for training. We inspect cases that were misclassified by this system and try to assess to what extent script knowledge should potentially be able to improve the results. This helps us to get an idea to which extent the data collection can fulfill its purpose to prove the applicability of script knowledge for natural language understanding. We find two reasons for the lack of contribution of script knowledge, which go back to the question collection procedure of MCScript: First, questions were collected irrespective of a concrete text but on the scenario only, which results in the problem that a number of questions has the same answer for any text within the scenario. Such answers can easily be memorized by the systems. Second, a number of questions ask about very general information in a scenario, requiring inference over more general commonsense knowledge, which is not supported by script knowledge.

We conclude from this that the MCScript corpus is appropriate for testing inference over a broader type of commonsense knowledge than script knowledge. This conclusion lead us to a repetition of the data collection process, in which we eliminated the aforementioned weaknesses (Chapter 10).

The structure of this chapter is as follows:

- In Section 9.1, we present baseline experiments on MCScript. We introduce several models and evaluate their performance on the data.
- Section 9.2 gives an overview of the systems that participated in the shared task based on MCScript. We briefly introduce and explain the participating systems and analyze their performance.
- Since we find that a range of systems performs well on MCScript without using script knowledge or even more general commonsense knowledge, we present a thorough analysis of our data in Section 9.3. This analysis reveals properties of the data collection process which delimit the usefulness of script knowledge for systems that are

tested on MCScript, but indicate that more general commonsense knowledge is required.

## 9.1 Baseline Experiments

In this section, we assess the performance of baseline models on MCScript, using accuracy as the evaluation measure. We employ models of differing complexity in order to assess the general difficulty of the data set: two unsupervised models using only word information and distributional information, respectively, and two supervised neural models. We assess performance on two dimensions: One, we show how well the models perform on text-based questions as compared to questions that require common sense for finding the correct answer. Two, we evaluate each model for each different question type.

### 9.1.1 Models

#### Word Matching Baseline

We first use a simple word matching baseline, by selecting the answer that has the highest literal overlap with the text. In case of a tie, we randomly select one of the answers.

#### Sliding Window

The second baseline is a sliding window approach that looks at windows of $w$ tokens on the text. Each text and each answer are represented as a sequence of word embeddings. The embeddings for each window of size $w$ and each answer are then averaged to derive window and answer representations, respectively. The answer with the lowest cosine distance to one of the windows of the text is then selected as correct.

#### Bilinear Model

We employ a simple neural model as a third baseline. In this model, each text, question, and answer is represented by a vector. For a given sequence of words $w_1 \dots w_n$, we compute this representation by averaging over the components of the word embeddings $\mathbf{w}_i$ that correspond to a word $w_i$, and then apply a linear transformation using a weight matrix. This

procedure is applied to each answer $a$ to derive an answer representation $\mathbf{a}$. The representation of a text $\mathbf{t}$ and of a question $\mathbf{q}$ are computed in the same way. We use different weight matrices for $\mathbf{a}$, $\mathbf{t}$ and $\mathbf{q}$, respectively. A combined representation $\mathbf{p}$ for the text–question pair is then constructed using a bilinear transformation matrix $\mathbf{W}$:

$$\mathbf{p} = \mathbf{t}^\top \mathbf{W} \mathbf{q} \tag{9.1}$$

We compute a score for each answer by using the dot product and pass the scores for both answers through a softmax layer for prediction. The probability $p$ for an answer $a$ to be correct is thus defined as:

$$p(a|t,q) = softmax(\mathbf{p}^\top \mathbf{a}) \tag{9.2}$$

**Attentive Reader**

The attentive reader is a well-established machine comprehension model that reaches good performance e.g. on the *CNN/Daily Mail* corpus (Hermann et al., 2015; Chen et al., 2016). We use the model formulation by Chen et al. (2016) and Lai et al. (2017), who employ bilinear weight functions to compute both attention and answer-text fit. Bi-directional GRUs are used to encode questions, texts and answers into hidden representations. For a question $q$ and an answer $a$, the last state of the GRUs, $\mathbf{q}$ and $\mathbf{a}$, are used as representations, while the text is encoded as a sequence of hidden states $\mathbf{t}_1 ... \mathbf{t}_n$. We then compute an attention score $s_j$ for each hidden state $\mathbf{t}_j$ using the question representation $\mathbf{q}$, a weight matrix $\mathbf{W}_a$, and an attention bias $b$. Last, a text representation $\mathbf{t}$ is computed as a weighted average of the hidden representations:

$$\begin{aligned} s_j &= softmax_j(\mathbf{t}_j^\top \mathbf{W}_a \mathbf{q} + b) \\ \mathbf{t} &= \sum_j s_j \mathbf{t}_j \end{aligned} \tag{9.3}$$

The probability $p$ of answer $a$ being correct is then predicted using another bilinear weight matrix $\mathbf{W}_s$, followed by an application of the softmax function over both answer options for the question:

$$p(a|t,q) = softmax(\mathbf{t}^\top \mathbf{W}_s \mathbf{a}) \tag{9.4}$$

#### Implementation Details

Texts, questions and answers were tokenized using *NLTK*[1] and lowercased. We used 100-dimensional *GloVe* vectors[2] (Pennington et al., 2014) to embed each token. For the neural models, the embeddings are used to initialize the token representations, and are refined during training. For the sliding similarity window approach, we set $w = 10$.

The vocabulary of the neural models was extracted from training and development data. For optimizing the bilinear model and the attentive reader, we used vanilla stochastic gradient descent with gradient clipping, if the norm of gradients exceeds 10. The size of the hidden layers was tuned to $64$, with a learning rate of $0.2$, for both models. We apply a dropout of $0.5$ to the word embeddings. Batch size was set to $25$ and all models were trained for 150 epochs. During training, we measured performance on the development set, and we selected the model from the best performing epoch for testing.

### 9.1.2 Results and Evaluation

#### Human Upper Bound

As an upper bound for model performance, we assess how well humans can solve our task. Two trained annotators labeled the correct answer on all instances of the test set. They agreed with the gold standard in 98.2 % of cases. This result shows that humans have no difficulty in finding the correct answer, irrespective of the question type.

#### Performance of the Baseline Models

Table 9.1 shows the performance of the baseline models as compared to the human upper bound and a random baseline. As can be seen, neural models have a clear advantage over the pure word overlap baseline, which performs worst, with an accuracy of $54.4\%$.

The low accuracy is mostly due to the nature of correct answers in our data: Each correct answer has a low overlap with the text by design. Since the overlap model selects the answer with a high overlap to the text, it does not perform well. In particular, this also explains

[1] http://www.nltk.org/

[2] https://nlp.stanford.edu/projects/glove/

| *Model* | *Text* | *CS* | *Total* |
|---|---|---|---|
| Chance | 50.0 | 50.0 | 50.0 |
| Word Overlap | 41.8 | 59.0 | 54.4 |
| Sliding Window | 55.7 | 53.1 | 55.0 |
| Bilinear Model | 69.8 | 71.4 | 70.2 |
| Attentive Reader | **70.9** | **75.2** | **72.0** |
| Human Performance | | | 98.2 |

**Table 9.1:** Accuracy of the baseline systems on text-based (*Text*), on commonsense-based questions (*CS*), and on the whole test set (*Total*). All numbers are percentages.

the very bad result on text-based questions. The sliding similarity window model does not outperform the simple word overlap model by a large margin: Distributional information alone is insufficient to handle complex questions in the dataset.

Both neural models outperform the unsupervised baselines by a large margin. When comparing the two models, the attentive reader is able to beat the bilinear model by only 1.8%. A possible explanation for this is that the attentive reader only attends to the text. Since many questions cannot be directly answered from the text, the attentive reader is not able to perform significantly better than a simpler neural model.

What is surprising is that the attentive reader works better on commonsense-based questions than on text questions. This can be explained by the fact that many commonsense questions do have prototypical answers within a scenario, irrespective of the text. The attentive reader is apparently able to just memorize these prototypical answers, thus achieving higher accuracy.

Inspecting attention values of the attentive reader, we found that in most cases, the model is unable to properly attend to the relevant parts of the text, even when the answer is literally given in the text. A possible explanation is that the model is confused by the large amount of questions that cannot be answered from the text directly, which might confound the computation of attention values.

Also, the attentive reader was originally constructed for reconstructing literal text spans

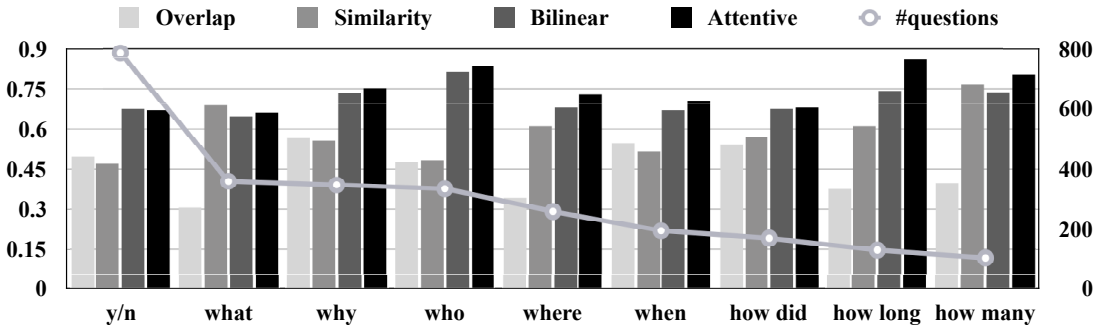



**Figure 9.1:** Accuracy values of the baseline models on question types appearing $> 25$ times.

as answers. Our mode of answer collection, however, results in many correct answers that cannot be found verbatim in the text. This presents difficulties for the attention mechanism.

The fact that an attention model outperforms a simple bilinear baseline only marginally shows that MCScript poses a new challenge to machine comprehension systems. Models concentrating solely on the text are insufficient to perform well on the data.

### Performance on Question Types

Figure 9.1 gives accuracy values of all baseline systems on the most frequent question types (appearing >25 times in the test data), as determined based on the question words (see Section 8.3). The numbers depicted on the left-hand side of the y-axis represent model accuracy. The right-hand side of the y-axis indicates the number of times a question type appears in the test data.

The neural models unsurprisingly outperform the other models in most cases, and the difference for *who* questions is largest. A large number of these questions ask for the narrator of the story, who is usually not mentioned literally in the text, since most stories are written in the first person.

It is also apparent that all models perform rather badly on *yes/no* questions. Each model basically compares the answer to some representation of the text. For yes/no questions, this makes sense for less than half of all cases. For the majority of yes/no questions, however, answers consist only of *yes* or *no*, without further content words.

## 9.2 Shared Task

In this section, we summarize the results of a shared task on the basis of the MCScript data. We briefly explain all systems that participated and discuss which techniques seemed to work well. In particular, we look at which kinds of commonsense knowledge were used and found to be beneficial.

### 9.2.1 Participants

In total, 11 teams participated in the shared task: 8 teams from China, and one team each from Spain, Russia and the US. Except for one team, all participating models rely on recurrent neural network techniques to encode texts, questions and/or answers. The one team that did not apply neural methods proposed an alternative approach based on clustering techniques and scoring word overlap.

Only 3 of the 11 teams made explicit use of commonsense knowledge: Two approaches used ConceptNet, either in the form of features extracted from ConceptNet relations or in the form of pretrained *Numberbatch* embeddings (Speer et al., 2017). One participating system made use of script knowledge in the form of event sequence descriptions.

Other resources commonly used by participants include pretrained word embeddings such as GloVe (Pennington et al., 2014) or word2vec (Mikolov et al., 2013a,b), and preprocessing pipelines such as NLTK[3]. In the following, we provide short summaries of the participants' systems and we give an overview of models and resources used by them (Table 9.2).

[3] https://www.nltk.org/

| Rank | Team name | Main model | Commonsense | Other resources | Acc. |
|---|---|---|---|---|---|
| 1 | Yuanfudao | LSTM | ConceptNet | GloVe, Wikipedia, POS and NE tagging | 0.84 |
| 2 | MITRE | LSTM | – | word2vec, Twitter, stemming | 0.82 |
| 3 | Jiangnan | LSTM | – | GloVe, CoVe, POS and NE tagging | 0.81 |
| 4 | ELiRF-UPV | LSTM | ConceptNet | – | 0.75 |
| 5 | YNU_Deep | LSTM | – | GloVe | 0.75 |
| 6 | ZMU | LSTM | – | word2vec, GloVe | 0.74 |
| 7 | ECNU | LSTM | – | GloVe | 0.73 |
| 8 | YNU_AI1799 | LSTM/CNN | – | word2vec, GloVe | 0.72 |
| 9 | YNU-HPCC | LSTM/CNN | – | word2vec | 0.71 |
| 10 | CSReader | LSTM | – | lemmatization, GloVe | 0.63 |
| 11 | IUCM | k-means | DeScript, MCScript[4] | NLTK | 0.61 |

**Table 9.2:** Overview of techniques and resourced used by the participating systems.

**Non-neural methods** *IUCM* (Reznikova and Derczynski, 2018) applied an unsupervised approach that assigns the correct answer to a question based on text overlap. Text overlap is computed based on the given passage and text sources of the same topic. Different clustering and topic modeling techniques are used to identify such text sources in MCScript and DeScript.

**Neural-network based models** Apart from *IUCM*, all participating systems are neural end-to-end models that employ recurrent and/or convolutional neural network architectures. Systems mainly differ with respect to details of the architecture and in the form of how words are represented.

*Yuanfudao* (Wang et al., 2018) applies a three-way attention mechanism to model interactions between the text, question and answers, on top of BiLSTMs. Each word in a text, question, and answer is represented by a vector of GloVe embeddings and additional information from part-of-speech tagging, name entity recognition, and relation extraction (based on ConceptNet). The model is pretrained on another large machine comprehension dataset, namely the RACE corpus.

*MITRE* (Merkhofer et al., 2018) use a combination of 3 systems - two LSTMs with attention mechanisms, and one logistic regression model using patterns based on the vocabulary of the training set. The two neural models use different word embeddings - one trained on GoogleNews, and the other trained on Twitter, which were enriched with word overlap features. Interestingly, the simple logistic regression model achieves competitive performance and would have ranked 4th as an individual system.

*Jiangnan* (Xia, 2018) applies a BiLSTM over GloVe and CoVe embeddings (McCann et al., 2017) with an additional attention mechanism. The attention mechanism computes soft word alignment between words in the question and the text or answer. Manual features, including part-of-speech tags, named entity types, and term frequencies, are employed to enrich word token representations.

*ELiRF-UPV* (González et al., 2018) employ a BiLSTM with attention for finding similarities between texts, questions, and answers. Each word is represented based on Numberbatch embeddings, which encode information from ConceptNet.

*YNU_Deep* (Ding and Zhou, 2018) test different LSTMs and BiLSTMs variants to encode questions, answers and texts. A simple attention mechanism is applied between question–answer and text–answer pairs. The final submission is an ensemble of five model instances.

*ZMU* (Li and Zhou, 2018) consider a wide variety of neural models, ranging from CNNs, LSTMs and BiLSTMs with attention, together with pretrained word2vec and GloVe embeddings. They also employ data augmentation methods typically used in image processing. Their best performing model is a BiLSTM with attention mechanism and combined GloVe and word2vec embeddings.

*ECNU* (Sheng et al., 2018) use BiGRUs and BiLSTMs to encode questions, answers and texts. They implement a multi-hop attention mechanism from question to text (a Gated Attention Reader (Dhingra et al., 2017)).

*YNU_AI1799* (Liu et al., 2018) submitted an ensemble of neural network models based on LSTMs, RNNs, and BiLSTM/CNN combinations, with attention mechanisms. In addition to word2vec embeddings, positional embeddings are used that are generated based on word embeddings.

*YNU-HPCC* (Yuan et al., 2018) use an ensemble of neural networks with stacked CNN and LSTM layers and attention.

*CSReader* (Jiang and Sun, 2018) use GRUs to encode questions and texts. Answer and text are combined by using an attention mechanism that models soft word alignments, inspired by work on Natural Language Inference (Bowman et al., 2015). Two answer classifiers based on these representations are ensembled for prediction.

### 9.2.2 Results

Tables 9.3 and 9.4 give detailed results for all participating systems. We performed pairwise significance tests using an approximate randomization test (Yeh, 2000) over texts. At an accuracy of 84%, the best participating team Yuanfudao performed significantly better ($p<0.05$) than the second best system, MITRE (82%).

| Rank | Team name | Total | Commonsense | Text | Out of Domain |
|---|---|---|---|---|---|
| 1 | Yuanfudao | **0.84*** | **0.82** | **0.85** | **0.79** |
| 2 | MITRE | 0.82 | 0.79 | 0.83* | 0.78 |
| 3 | Jiangnan | 0.81* | 0.80 | 0.81* | 0.75* |
| 4 | ELiRF-UVP | 0.75 | **0.82** | 0.73 | 0.70 |
| 5 | YNU_Deep | 0.75 | 0.79 | 0.73 | 0.66 |
| 6 | ZMU | 0.74 | 0.80 | 0.72 | 0.66 |
| 7 | ECNU | 0.73 | 0.77 | 0.72 | 0.69 |
| 8 | YNU_AI1799 | 0.72 | 0.76 | 0.71 | 0.67 |
| 9 | YNU_HPCC | 0.71* | 0.78* | 0.69* | 0.64* |
| 10 | CSReader | 0.63 | 0.64* | 0.63 | 0.59 |
| 11 | IUCM | 0.61 | 0.54 | 0.64 | 0.58 |
| – | Attentive Reader | 0.72 | 0.75 | 0.71 | 0.69 |
| – | Sliding Window | 0.55 | 0.53 | 0.56 | 0.52 |
| – | Human Performance | 0.98 | | | |

**Table 9.3:** The accuracy of participating systems and the two baselines in total, on commonsense-based questions, text-based questions and on out-of-domain questions (from the 5 held-out testing scenarios). The best performance for each column is marked in **bold print**. Significant differences in results between two adjacent lines are marked by an asterisk (* $p<0.05$) in the upper line. The last line shows the human upper bound on MCScript as comparison.

| Rank | Team name | y/n | what | why | who | where | when |
|---|---|---|---|---|---|---|---|
| 1 | Yuanfudao | **0.76** | **0.87** | **0.85** | **0.93** | **0.88** | 0.81 |
| 2 | MITRE | **0.76** | 0.83 | 0.82 | 0.91 | 0.85 | 0.77 |
| 3 | Jiangnan | 0.75* | 0.80* | 0.80 | 0.88 | 0.84* | **0.82*** |
| 4 | ELiRF-UVP | 0.72 | 0.68 | 0.79 | 0.86 | 0.69 | 0.74 |
| 5 | YNU_Deep | 0.73 | 0.66 | 0.75 | 0.86 | 0.71 | 0.72 |
| 6 | ZMU | 0.73 | 0.65 | 0.77 | 0.81 | 0.72 | 0.75 |
| 7 | ECNU | 0.71 | 0.66 | 0.75 | 0.82 | 0.73 | 0.68 |
| 8 | YNU_AI1799 | 0.70 | 0.68 | 0.78* | 0.80 | 0.67 | 0.72 |
| 9 | YNU_HPCC | 0.72* | 0.66* | 0.71 | 0.83* | 0.65 | 0.66 |
| 10 | CSReader | 0.54 | 0.59* | 0.68 | 0.76* | 0.62* | 0.63 |
| 11 | IUCM | 0.54 | 0.75 | 0.66 | 0.45 | 0.77 | 0.61 |
| – | Attentive Reader | 0.67 | 0.66 | 0.75 | 0.84 | 0.73 | 0.71 |
| – | Sliding Window | 0.47 | 0.69 | 0.56 | 0.48 | 0.61 | 0.51 |

**Table 9.4:** Accuracy of participating systems and the baselines on the six most frequent question types. The best performance for each column is marked in **bold print**. Significant differences in results between two adjacent lines are marked by an asterisk (* $p<0.05$) in the upper line.

Except for *when* questions, Yuanfudao also achieved the best performance at each question type. However, individual differences in results over the 2nd place system were not found to be significant. The top three participating teams, Yuanfudao, MITRE and Jiangnan, all significantly outperform the remaining teams on text-based questions (>80% vs. <74%) as well as on *yes/no*, *what*, *where* and *when* questions.

In comparison to our baselines, all teams but IUCM significantly outperform Sliding Window. Results of the Attentive Reader are in line with those of the participating systems ranked 7–9: ECNU, YNU_AI1799 and YNU_HPCC. The six top-ranked systems all significantly outperform both of our baselines. On out-of-domain questions, only the top 3 performing models significantly outperform the Attentive Reader baseline, while all models significantly outperform the Sliding Window approach.

For commonsense-based questions as well as for questions on *why* and *who*, results are considerably less consistent: While the top ranked system significantly outperforms teams ranked 7th or lower, most pairwise differences between the top teams are not statistically significant. This implies that the set of correctly answered questions considerably varies between systems, either due to randomness or because they excel at different inference problems.

We found that 19.3 % of the questions in the test set were answered correctly by each participating system. These questions mainly contain text-based questions with an answer that is literally given in the text. Also, there are many commonsense-based questions with a standardized correct answer, as shown in Example 20. Only few of the stories in MCScript cover a long time span, so the answer to such questions is always similar.

(20) **Q:** How long did it take to pump up the tires?
a. just a few minutes b. a few hours

In contrast, only 1% of questions could not be answered by any of the participating models. Answering these questions mainly requires complicated inference steps, such as counting or plausibility judgments.

### 9.2.3 Discussion

We briefly highlight some of the findings by the shared task participants.

**Commonsense knowledge sources.** One of the main goals of this shared task was to provide an extrinsic evaluation framework for models of commonsense knowledge. However, only three participants actually made use of resources of commonsense knowledge.

Most prominent is the use of ConceptNet, a large-scale knowledge graph that is built from several handcrafted and crowdsourced sources. It was employed by two of the top 5 scoring models: Yuanfudao use it to learn their own ConceptNet-based relation embeddings. ELiRF-UPV make use of Numberbatch word embeddings, which are learned based on ConceptNet data. Ablation analyses conducted by Yuanfudao indicate that the addition of ConceptNet increases overall accuracy by almost 1% absolute.

In contrast, only one participant used crowdsourced script data from the DeScript corpus in their final submission, IUCM. They found that the use of script data, instead of or in addition to texts, improved performance by up to 0.3% absolute. CSReader tried to extend their neural model with script data from OMCS, but report that it did not result in an improvement. These results indicate that enhancing models with script knowledge seems to only marginally influence the results. In Section 9.3, we take a closer look at this observation and try to identify reasons for the limited usefulness of script knowledge.

No participant made use of narrative chains or other forms of structured/learned representations of scripts or events (such as event embeddings).

**Pretraining.** Most participants made use of pretraining in the form of word embeddings such as word2vec or GloVe, that were build on large data collections. Yuanfudao used the RACE dataset, which is the largest available multiple-choice machine comprehension corpus, for pretraining the complete model for several epochs. In their ablation analysis, they found pretraining to have the largest effect on model performance, with improvements in accuracy of up to 1.4% absolute. This result underlines that the comparably small size of MCScript naturally restricts how much neural approaches can learn from the data without overfitting.

**Word representations.** For representing tokens, most participants used word2vec embeddings, GloVe embeddings, or combinations thereof. The participating teams used different dimensionality sizes, and some of them refitted the vectors while others did not, leading to differing outcomes for both embedding types. In summary, none of the two representations seems to clearly outperform the other.

In contrast, participants consistently found that extending word representations with additional features improves results: For example, Yuanfudao and Jiangnan use predicted part-of-speech tags and named entity information, as well as term frequency, and report improvements of up to 1% absolute in accuracy. Also, some participants report the use of word overlap features. Most notably, MITRE found that a logistic regression classifier based on overlap features can achieve performance competitive with neural approaches.

In general, additional features seem to be beneficial, since they provide more explicit or

additional information that can be leveraged by neural networks and other classifiers.

**Preprocessing.** Several participants reported that lemmatizing and stop word removal further improved their results. A prominent example is the submission by MITRE, who use a stemmer to derive root forms for all words, in order to compute overlap and co-occurrence statistics between answers and text/questions.

## 9.3 Assessing the Contribution of Script Knowledge

Important findings of the last two sections were that (1) commonsense knowledge (and particularly, script knowledge) are not necessarily required to perform well on MCScript and (2) script-based questions are not necessarily more difficult for a system than text-based questions. In this section, we try to find the reasons for these two unexpected features of the data. To do so, we take a closer look at one of the participating systems. Although initially planned as a baseline system by one of the participants (Merkhofer et al., 2018), it reached the top 5, outperforming most neural models. Notably, no commonsense or script knowledge is used in the model, and all features are based on the overlap of answer, question and text only.

We choose to analyze the results of this model, because it provides an interesting analysis opportunity: The model is a logistic regression classifier, so it is possible to look at the feature weights that are learned during training and to directly interpret the importance of specific features, which is a lot harder with neural models. Also, in spite of its simplicity, the model is able to handle many questions that we expected to require commonsense inference. To get an additional estimate of how much using script knowledge can potentially help to answer questions, we take a closer look at the questions which are not correctly classified by the system.

We find that for most of these cases, using script knowledge as additional resource would help only marginally if at all for the classification. Most questions for which one would expect script knowledge to be beneficial are already classified correctly without the usage of additional resources. The remaining questions require inference over more general commonsense knowledge.

### 9.3.1 Logistic Model

The benchmark model proposed by Merkhofer et al. (2018) is a logistic regression classifier using 3 types of features:

- **Length features (6 features).** These features numerically encode the length of text, answer and questions on word and character level, encoding for example the information whether longer or shorter answers are in general favorable.
- **Overlap features (3 features).** Similar to the overlap baseline presented in Section 9.1, they encode the amount of overlap between the elements of the data set. The three features encode the token overlap between (1) the text and the answer, (2) the question and the answer, and (3) the text and the question. While the first two features encode information about whether the answer is mentioned in the text or question, the third feature should mainly be beneficial for decision questions, checking whether the question is supported by material in the text, or not.
- **Lexical patterns. (108,900 features)** The large majority of features that are used are binary patterns that encode the presence or absence of words or combinations of words in answer, text and question. These pattern features are a very explicit way of encoding content. The first group of features encode the presence of all possible words in the answer (3,400 features). The second group encodes the presence of words that are both in text and answer (2,000 features). The third, largest group of features encodes possible combinations of words in question and answer (103,500 features).

During training, each text-question-answer pair is treated as one single training instance, i.e. correct and incorrect answer are treated separately. During testing time, the answer with the higher probability of being correct is taken as the correct answer. All texts, answers and questions are stemmed using the Porter stemmer (Porter, 1980), and articles are removed. *LibLinear* (Fan et al., 2008) is used for training the regression classifier. A bias term and L2 regularization with standard parameters were used for training.

| | $\sigma\|w\|$ | $w$ | story | question | answer |
|---|---|---|---|---|---|
| 1 | 0.61 | 1.85 | | | ye |
| 2 | 0.50 | -1.89 | it | | it |
| 3 | 0.45 | -2.22 | they | | they |
| 4 | 0.35 | 1.00 | | | they |
| 5 | 0.34 | -1.64 | wa | | wa |
| 6 | 0.33 | -1.42 | in | | in |
| 7 | 0.28 | -2.25 | at | | at |
| 8 | 0.27 | -1.97 | | | friend |
| 9 | 0.26 | 1.98 | | | narrat |
| 10 | 0.24 | -1.66 | for | | for |

| | $\sigma\|w\|$ | $w$ | description |
|---|---|---|---|
| 1 | 1.99 | 1.2814 | $\|S \cap A\|$ |
| 2 | 1.18 | -0.5591 | answer length (words) |
| 3 | 0.49 | 0.0394 | answer length (chars) |
| 4 | 0.15 | -0.1728 | $\|Q \cap A\|$ |
| 5 | 0.11 | -0.0005 | story length (chars) |
| 6 | 0.09 | 0.0026 | story length (words) |
| 7 | 0.04 | 0.0041 | question length (chars) |
| 8 | 0.03 | -0.0219 | $\|S \cap Q\|$ |
| 9 | 0.02 | 0.0114 | question length (words) |

**Figure 9.2:** The importance of features, with the top 10 lexicalized patterns on the left and overlap/length features on the right. $w$ is the actual feature weight. The product of the weights magnitude with the standard deviation ($\sigma|w|$) balances rare and more common features and is used for ranking the features. Both tables taken from Merkhofer et al. (2018).

### 9.3.2 Discussion

**Importance of Features.** The tables in figure 9.2 (taken from Merkhofer et al. (2018)) illustrate which features were found to be most important for the classifier. The left table indicates that the classifier has learned that answers starting with *ye(s)* are very likely to be true, as well as answers that contain the words *friend*, *narrat(or)* or *they*. The ranking in the right table indicates that answer length is a strong indicator for the correctness of an answer.

Examples 21 and 22 give 2 texts and questions as illustrating examples. Both questions ask about the protagonist, so the correct answer is in both cases "the narrator". We found many such cases in the data, which are easy to answer for a system.

(21) *Text*: (...) I use a bread knife which makes cutting it easier, and cut off just what I need for the toast, leaving the rest whole. I wrap and store the remaining loaf for next time. (...) I unplug it immediately but let it cool down before putting it away as it is very hot. I butter the bread using a knife and then put away the knives and butter.

*Question*: Who is toasting the bread?

(22) *Text*: (...) So, we decided to order a pizza. I looked in the telephone book and found several pizza restaurants that would deliver. We all talked it over and settled on a

pizza from Domino's. Then we discussed what we would like on our pizza and what size we should get. (...)
*Question*: Who ordered the pizza?

Examples 23 and 24 show other cases of standardized answers. For both questions, the correct answer would be "a few minutes", which appears often in the data. Although both questions were labeled as commonsense-based, the correct answer is identical across different texts, which makes it very easy for systems to memorize it.

(23) *Text*: My family and I decided that the evening was beautiful so we wanted to have a bonfire. First, my husband went to our shed and gathered some dry wood. I placed camp chairs around our fire pit. Then my husband placed the dry wood in a pyramid shape inside the fire pit. (...) We ate our S'mores as we joked, laughed and told stories around our beautiful fire. (...)
*Question*: How long did it take to build the fire?

(24) *Text*: Last night after dinner, I had "dishwasher duty"; That means that it was my turn to do the dishes and unload the dishwasher. After the dishwasher had washed the dishes, I first opened the door to the dishwasher. I unloaded the bottom rack first. That way, if any water drips or leftover food were stuck on the dishes still, they wouldn't trickle down to my clean dishes. The bottom drawer usually has my plates in it. (...)
*Question*: How long did it take them to unload the dishwasher?

These findings indicate that models are able to exploit special features of the data in order to find the correct answer. The data seems to be skewed in a way such that answers that talk about the narrator, i.e. the protagonist of the story, are more likely to be true. The most probable reason for this skewness lies in the way of question collection. Since the questions were not collected based on stories, but rather on a very general description of a scenario, the same type of question appears frequently across different participants and especially across scenarios. Questions about the main acting person in the scenario, i.e. the protagonist, seem to be one such problematic type, because they are asked across all scenarios: It is an obvious choice to ask for the protagonist in any scenario, such as *Who planted the tree?* in the PLANTING A TREE scenario, or *Who took the train?* in the TAKING A TRAIN SCENARIO. Also, questions about

central events (*Who dug the hole?* or *Who entered the train?*) in the scenarios usually have the same answer, irrespective of a text.

Also, the fact that decision questions are often answered with *yes* does not come surprising: The turkers tend to ask for events in the scenario, knowing that they must have happened (*Did they dig a hole?* or *Did they pay the fare?*).

**Possible Improvements through Script Knowledge.** In order to estimate how much script knowledge could improve the classification, we sampled 50 questions from the development set that were misclassified by the logistic model and manually looked for cases where participant or event information could help. We found that in approximately 10% of cases, script knowledge could possibly improve the classification.

25 shows one such example, where script knowledge could help in finding that in the VISITING A SAUNA script, people usually sit on benches rather than on the floor. Similarly, in Example 26, knowledge about the PLANTING A TREE script would help in finding that saplings are usually used rather than old trees.

(25) *Text*: It was a long day at work and I decided to stop at the gym before going home. I ran on the treadmill and lifted some weights. I decided I would also swim a few laps in the pool. Once I was done working out, I went in the locker room and stripped down and wrapped myself in a towel. I went into the sauna and turned on the heat. I let it get nice and steamy. I sat down and relaxed. I let my mind think about nothing but peaceful, happy thoughts. I stayed in there for only about ten minutes because it was so hot and steamy. When I got out, I turned the sauna off to save energy and took a cool shower. I got out of the shower and dried off. After that, I put on my extra set of clean clothes I brought with me, and got in my car and drove home.
*Question*: Where did they sit inside the sauna?
*Correct Answer:* on a bench
*Incorrect Answer:* on the floor

(26) *Text*: Before you plant a tree, you must contact the utility company. They will come to your property and mark out utility lines. Without doing this, you may dig down and hit a line, which can be lethal! Once you know where to dig, select what type of tree you want. Take things into consideration such as how much sun it gets, what

zone you are in, and how quickly you want it to grow. Dig a hole large enough for the tree and roots. Place the tree in the hole and then fill the hole back up with dirt. Many small trees will need support until they are bigger. This will ensure they grow tall and straight. It is crucial to keep the tree watered! It will need a lot of water at first. Mulch is helpful to keep the soil moist. It's also important to wait until the threat of a frost has passed or is not in the near future. Giving care to your new tree will ensure that it will grow big and strong!

*Question*: Were they saplings or old trees?

*Correct Answer:* saplings

*Incorrect Answer:* old trees

(27) *Question*: Would it have been easier to plant the tree if they had help?

*Correct Answer:* yes

*Incorrect Answer:* no

Cases such as Example 25 and 26 were rare and hard to find in the data. To the largest extent, the misclassified questions would require generic inference over a more general type of commonsense knowledge that is typically not part of script knowledge.

Examples 27 and 28 illustrate this: The simple fact that planting a tree gets easier if you have help is hard to infer for a machine, as well as the fact that one takes driving lessons in order to get a driver's license.

This problem can also be attributed to the mode of question collection: Turkers were not guided to ask about participants and events of a story specifically, which would result in a better focus of the questions on script knowledge. Instead, many questions are asked in a very open way, and do not address information that is encoded in scripts.

(28) *Text*: My mom is teaching me to drive. First thing, we check to make sure the blinkers are working. I flipped the lever on the left side of the steering column up for the right blinker and down for the left. Both arrows under the speedometer blinked and on the outside of the car in the front and back. Next we got in and fastened our seat belts, then I checked the side mirrors and rear view to make sure I could see. I inserted the keys into the ignition and using my thumb and first finger pushed the switch forward. I checked the mirrors again to make sure all people, animals, and objects

where out of the way. I pulled the gear shift towards me and pulled down one click going from P (park) to R (reverse). I check the mirrors once more and turned my head to the right to look over my shoulder to back out of the driveway. At the end of the driveway, I pushed the turn signal lever down to signal I was turning left. I looked to the right and to the left to check for traffic twice and pulled out.

*Question*: Why did they want to take driving lessons?

*Correct Answer:* To get their license.

*Incorrect Answer:* To apply for a job.

To alleviate these problems, we created a second corpus, based on a revised version of the question collection experiment, as described in Chapter 10.

## 9.4 Conclusion

In this chapter, we described baselines models to assess the general difficulty of our MCScript corpus. We also reported on the results of a SemEval shared task that we conducted based on the corpus.

We find that baselines models struggle on the data, while the task is trivial for humans, illustrating that the task is generally challenging. Surprisingly, script-based questions are generally not harder to answer than text-based questions for our baseline models. While we expected that models which have access to script knowledge should perform better on the data, no participant of the shared task found notable effects when including script knowledge data into their models. The best-performing system instead made use of ConceptNet which lead to small but notable improvements.

Since two of our main expectations were not fulfilled, namely that script knowledge should help systems, and that script-based questions are harder to answer, we decided to perform a more rigorous investigation of the performance of one of the best performing models in the shared task. Merkhofer et al. (2018) use a simple logistic model that makes no use of commonsense knowledge, but performs very well on our data. By an analysis of the model's output and feature weights, as well as a manual inspection of the data, we identified two main problems: (1) the answers to many script-based questions can be memorized by sys-

tems based on the training data and (2) many of the remaining questions require inference over more generic commonsense knowledge as encoded by scripts.

Our analysis yields an instructive opportunity for a revision of the data collection process. Based on such a revision, we repeated the data collection and created a second dataset, as described in Chapter 10.

Chapter 10

# MCScript2.0: A Revised Machine Comprehension Dataset

An important finding in Chapter 9 was that script knowledge is not necessarily required for performing well on MCScript.

In short, we found that this can be attributed to the way in which questions were collected. During the collection process, workers were not shown a text, but only a very short description of the scenario that the text refers to. As a result, many questions ask about general aspects of the scenario, without referring to actual details. This has the effect that there are many questions with standardized answers, i.e. questions that can be answered irrespective of a concrete reading text. Merkhofer et al. (2018) found that such information can essentially be learned from only the training data without external background information, using a simple logistic regression classifier and surface features regarding words in the text, question and answer candidates. Also, many questions require inference over general commonsense knowledge rather than script knowledge.

In this chapter, we present a new corpus, based on a new data collection method that results in a larger number of challenging questions that require script knowledge. In particular, we define a revised question collection procedure, which ensures a large proportion of commonsense questions, whose answering requires non-trivial inference. In the new question

| | |
|---|---|
| **T** | (...) We put our ingredients together to make sure they were at the right temperature, preheated the oven, and pulled out the proper utensils. We then prepared the batter using eggs and some other materials we purchased and then poured them into a pan. After baking the cake in the oven for the time the recipe told us to, we then double checked to make sure it was done by pushing a knife into the center. We saw some crumbs sticking to the knife when we pulled it out so we knew it was ready to eat ! |
| **Q1** | *When did they put the pan in the oven and bake it according to the instructions?* |
| | After eating the cake. ✗ |
| | After mixing the batter. ✓ |
| **Q2** | *What did they put in the oven?* |
| | The cake mix. ✓ |
| | Utensils. ✗ |

**Figure 10.1:** Example text fragment from MCScript2.0

collection procedure, we focus exclusively on questions about participants and events. The questions are formulated based on a concrete text, and the sentence containing the information is subsequently hidden in the text, such that for finding an answer, the required information needs to be inferred from background knowledge.

Based on the new method, we create *MCScript2.0*, a reading comprehension corpus focused on script events and participants. It contains more than 3,400 texts about everyday scenarios, together with more than 19,000 multiple-choice questions on these texts. All data were collected via crowdsourcing. Based on a crowdsourced reasoning type annotation, we find that about half of the questions require the use of commonsense and script knowledge for finding the correct answer, a notably higher number than in MCScript. Moreover, we show that in comparison to MCScript, commonsense-based questions in MCScript2.0 are also harder to answer, even for a system that makes use of a commonsense database.

Figure 10.1 shows a text snippet from a text in MCScript2.0, together with two questions with answer alternatives. To find an answer for question 1, information about the temporal order of the steps for baking a cake is required: The cake is put in the oven after mixing the batter, and not after eating it—a piece of information not given in the text, since the

event of putting the cake in the oven is not explicitly mentioned. Similarly, one needs script knowledge about which participants are typically involved in which events to know that the cake mix rather than the utensils is put into the oven. Both incorrect answer candidates are distractors: The utensils as well as the action of eating the cake are mentioned in the text, but incorrect answers to the question.

The structure of this chapter is as follows:

- In Section 10.1, we present a new collecting method for challenging questions whose answers require commonsense knowledge and in particular script knowledge, as well as a new machine comprehension dataset that was created with this method.
- We compare MCScript2.0 to MCScript, our first machine comprehension resource for evaluating models of script knowledge. We show that in comparison to MCScript, the number of questions that require script knowledge is increased by a large margin (Section 10.2).
- In Section 10.3, we show that the task is simple for humans, but that existing benchmark models, including a top-scoring machine comprehension model that utilizes a resource for commonsense knowledge, struggle on the questions in MCScript2.0. This holds in particular for questions that require commonsense knowledge, which are most difficult. Consequently, we argue that our dataset provides a more robust basis for future research on text understanding models based on script knowledge.

## 10.1 Corpus Creation

Texts, questions, and answer candidates are required for a multiple choice machine comprehension dataset. Our data collection process for texts and answers is based on the MCScript data and the methods developed there, but with several crucial differences. We create the data set via crowdsourcing, as for the creation of MCScript (Chapter 8). The question collection is revised to account for the shortcomings found with MCScript.

Similarly to MCScript, we are interested in questions that require inference over script knowledge for finding a correct answer. Creating such questions is challenging: When questions are collected by showing a reading text and asking crowdsourcing workers to write

questions, their answer can usually be read off the text. For MCScript, we thus decided to not show a reading text at all, but only a short summary of the text scenario. This resulted in too general questions, so we decided for a third option: We identified a number of *target sentences* in the reading text and guided workers to formulate questions about script-related details in these sentences. The target sentences were then hidden from the text, meaning that relevant information would have to be inferred from common sense during the answer collection and also in the task itself. In the following sections, we describe the three data collection steps in detail.

### 10.1.1 Text Collection

As a starting point, we reused all texts from MCScript (2,119 texts on 110 scenarios) for our data set. To increase the topical coverage and diversity, we added texts for 90 new scenarios to our collection. As for MCScript, we selected topically different and plausible everyday scenarios of varying complexity, which were not too fine-grained (such as *opening a window*). The scenarios were taken from 3 sources: First, we extracted scenarios from several script collections (Wanzare et al., 2016; Regneri et al., 2010; Singh et al., 2002) that are not part of MCScript. Second, we inspected the *Spinn3r* blog story corpus (Burton et al., 2009), a large corpus of narrative blog stories and identified additional scenarios in these stories. Third, we added new scenarios that are related to existing ones or that extend them.

We collected 20 texts per new scenario, using the same text collection method as for MCScript: We asked workers to tell a story about a certain everyday scenario "as if talking to a child". This instruction ensures that the resulting stories are simple in language and clearly structured. Texts collected this way have been found to explicitly mention many script events and participants (s. Chapter 4 and Chapter 8). They are thus ideal to evaluate script-based inference.

### 10.1.2 Question Collection

For the question collection, we follow the creation of MCScript in telling workers that the data are collected for a reading comprehension task for children, in order to get linguistically simple and explicit questions. However, as mentioned above, we guide workers towards

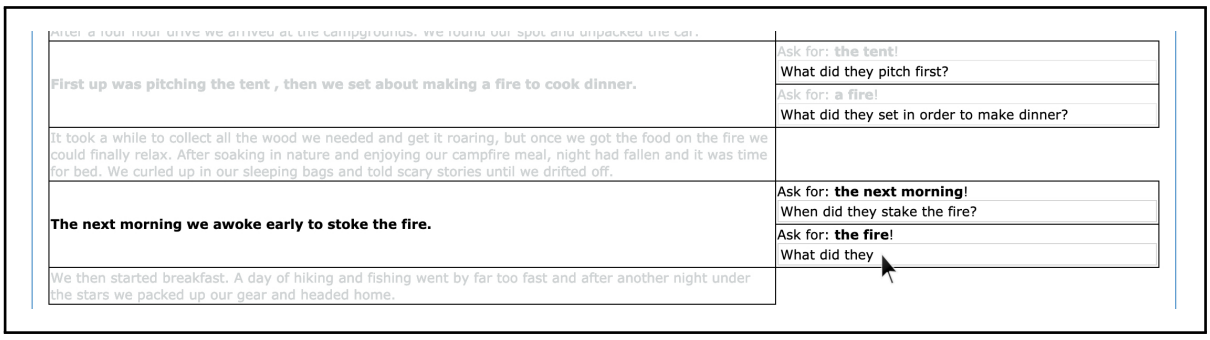



**Figure 10.2:** Screenshot of an item in the participant question collection experiment.

asking questions about target sentences rather than a complete text.

As target sentences, we selected every fourth sentence in a text. In order to avoid selecting target sentences with too much or too little content, we only considered sentences with less than 20 tokens, but that contained 2 or more noun phrases.[1]

In a series of pilot studies, we showed the texts with highlighted target sentences to workers and asked them to write questions about these sentences. We however found, that in many cases, the written questions were too general or nonsensical.

We concluded that an even more structured task was required and decided to concentrate on questions of two types: (1) questions that ask about participants, and (2) questions about the temporal event structure of a scenario. Participants are usually instantiated by noun phrases (NPs), while events are described by verb phrases (VPs). We thus used *Stanford CoreNLP* (Manning et al., 2014) to extract both NPs and VPs in the target sentences and split up the experiment into two parts: In the first part, workers were required to write questions that *ask about the given noun phrase*. Figure 10.2 shows a screenshot of an item from the first part. The first column shows the reading text with the target sentence highlighted. The second columns shows all extracted phrases with a field for one question per phrase.[2] Full details of the experiment instructions are given in the Supplemental Material.

In the second part, we asked workers to write a temporal question (when, how long, etc.) *us-*

[1] All parameters were selected empirically, by testing different values and analyzing samples of the resulting data.

[2] If the noun phrase was part of a prepositional phrase or a construction of the form "NP *of* NP", we took the whole phrase instead, because it is more natural to ask for the complete phrase. In order to avoid redundancy, we only looked at NPs that had no other NPs as parents. We also excluded noun phrases that referred to the narrator (*I*, *me* etc.).

*ing the given verb phrase*. We found that an exact repetition of the NP instructions for the second part ("ask about the given verb phrase") resulted in unnatural questions, so we adapted the instructions. A screenshot of the VP experiment is given in the Supplemental Material.

We showed each text to two workers and asked them to write one question per VP or NP. Workers were only allowed to work on either the VP or the NP part, since the instructions could easily be confused. In order to exclude yes/no questions, we did not accept inputs starting with an auxiliary or modal verb. Also, all questions needed to contain at least 4 words. We asked workers to use *they* to refer to the protagonist of the story and other types of mentions (e.g. pronouns like *I*, *you*, *we* or the word *narrator*) were not accepted.

### 10.1.3 Answer Collection

For collecting answer candidates we hid the target sentences from the texts and showed them with up to 12 questions, to keep the workload at an acceptable level. If there were more questions for a text, we selected 12 questions at random.

Since the target sentences are hidden in the texts, it can be expected that some questions cannot be answered from the text anymore. However, the necessary information for finding an answer might be inferred from script knowledge, so workers were explicitly told that they might need commonsense to find an answer. Some answers can still be read off the text, if other parts of the texts contain the same information as the hidden target sentences. For other questions, neither the text nor script knowledge provides sufficient information for finding an answer.

As for the creation of MCScript, workers first had to conduct a 4-way classification for each question to account for these cases: *text-based* (answer is in the text), *script-based* (answer can be inferred from script knowledge), *unfitting* (question doesn't make sense), *unknown* (answer is not known). Having such class annotations is not only useful for evaluation, but it also sensitizes workers for the fact that they are explicitly allowed to use background knowledge.

In the experiment, workers were also instructed to write both a correct and a plausible incorrect answer for questions labeled as *text-based* of *script-based*. We follow the creation of MCScript and require workers to write an alternative question if the labels *unfitting* or *unknown* are used, in order to level out the workload.

We presented each question to 5 workers, resulting in 5 judgments and up to 5 incorrect and correct answer candidates per question. For the final data set, we considered questions with a majority vote (3 out of 5) on *text-based* or *script-based*. We also included questions without a majority vote, but for which at least 3 workers assigned one of *text-based* or *script-based*. In that case, we assigned the new label *text-or-script* and also accepted the question for the final data set. This seemed reasonable, since at least 3 workers wrote answers for the question, meaning it could still be used in the final data collection. The remaining questions were discarded.

### 10.1.4 Answer Candidate Selection

In a last step, we selected one correct and one incorrect answer from all possible candidates per question for the data set. To choose the most plausible correct answer candidate, we adapt the procedure from MCScript: We normalize all correct answers (lowercasing, normalizing numbers[3], deleting stopwords[4]) and merge candidates that are contained in another candidate, and candidate pairs with a Levenshtein (1966) distance of less than 3. The most frequent candidate is selected as correct answer. If there was no clear majority, we selected a candidate at random.

To select an incorrect answer candidate, we adapt the *adversarial filtering* algorithm from Zellers et al. (2018). Our implementation uses a simple classifier that utilizes shallow surface features. The algorithm selects the incorrect answer candidate from the set of possible candidates that is most difficult for the classifier, i.e. an incorrect answer that is hard to tell apart from the correct answer (e.g. the incorrect answers in Figure 10.1: *eating* and *utensils* are also mentioned in the text). By picking incorrect answers with the adversarial filtering method, the dataset becomes robust against surface-oriented methods.

Practically, the algorithm starts with a random assignment, i.e. a random incorrect answer candidate per question. This assignment is refined iteratively, such that the most difficult candidate is selected. In each iteration, the algorithm splits the data into a random training part and a test part. The classifier is trained on the training part and then used to classify *all* possible candidates in the test part. The assignment of answer candidates in the test

[3] We used *text2num*, `https://github.com/ghewgill/text2num`.

[4] *and, or, to, the, a*

data is then changed such that the most difficult incorrect answer candidate per question is picked as incorrect answer. After several iterations through the entire dataset, the number of changed answer candidates usually stagnates and the algorithm converges.

For MCScript2.0, we use the logistic classifier mentioned in Section 7.1, which only uses surface features and is thus well suited for the filtering algorithm.

**Formalization**

**Data:** data set $D$, a randomly initialized assignment $S$, and a classifier $C$

**Result:** $\hat{S}$

**repeat**

  split the data into test batches of size $b$, such that each batch contains all questions for $b$ texts;

  **for** $D_{test}$ in batches **do**

    $D_{train} \longleftarrow D \backslash D_{test}$;

    $\mathcal{D}_{train} \longleftarrow compile(D_{train})$;

    train $C$ on $\mathcal{D}_{train}$;

    **for** all instances $< T_i, Q_i, a_i^+, < a_{i,0}^-...a_{i,j}^- >$ in $D_{test}$ **do**

      use $C$ to classify all incorrect answer candidates $a_{i,0}^-...a_{i,j}^-$;

      set $s_{i,j}$ to the index of the answer candidate with the highest probability of being correct;

    **end**

  **end**

**until** number of changed assignments stagnates or increases;

**Algorithm 1:** Adversarial Filtering for MCScript2.0

Algorithm 1 gives pseudo-code for the adversarial filtering algorithm on MCScript2.0. Formally, let a dataset be defined as a list of tuples $\langle t_i, q_i, a_i^+, \langle a_{i,0}^-...a_{i,j}^- \rangle \rangle$, where $t_i$ is a reading text, $q_i$ is a question on the text, $a_i^+$ is the correct answer (as selected via majority vote, s. Section 10.1.4) and $\langle a_{i,0}^-...a_{i,j}^- \rangle$ is a list of 3 to 5 incorrect answer candidates[5]. The aim of the algorithm is to find an assignment $\hat{S} = \{s_{0,0}...s_{i,j}\}$, where each $s_{i,j}$ is the index of the most difficult answer candidate in $\langle a_{i,0}^-...a_{i,j}^- \rangle$.

[5]Note that since there are several questions per text, the value of $t_i$ may appear in several instances.

| *text-based* | *script-based* | *text-or-script* | *unfitting* | *unknown* |
|---|---|---|---|---|
| 9,357 | 12,433 | 2,403 | 3,240 | 6,457 |
| **total answerable:** 24,193 | | | **total not answerable:** 9,397 | |

**Table 10.1:** Distribution of question labels, before validation.

A dataset that is *compiled* with the assignment $S$ is a list of instances $< t_i, q_i, a_i^+, a_i^- >$, such that there is only one incorrect answer candidate per question, according to the indices given by $S$.

Once the algorithm converges, $\hat{S}$ is used to compile the final version of the dataset, $\hat{\mathcal{D}}$, which contains incorrect answer candidates that are most likely to be correct.

For the batch size we tried values in $\{50, 100, 250, 500\}$, but we found that for all values, the performance of the classifier would drop close to chance level after only one iteration. We set $b = 250$, since the performance was closest to chance after convergence with that setting. Also, we defined that the algorithm converges if the number of changed assignments since the last iteration is $\leq 50$.

## 10.2 Corpus Analysis

### 10.2.1 General Statistics

In total, MCScript2.0 comprises 19,821 questions on 3,487 texts, i.e. 5.7 questions on average per text. The average length of texts, questions and answers is 164.4 tokens, 8.2 tokens and 3.4 tokens, respectively. In the data collection process, we crowdsourced 1,800 new texts, resulting in a total of 3,919 texts for 200 scenarios. On average, there are 1.98 target sentences per text. In the question collection, we gathered 42,132 questions that were used for the answer collection. For 8,242 questions, there was no clear majority on the question label. Table 10.1 shows the label distribution on the remaining 33,890 questions. 24,193 of these could be answered, i.e. 71%.

To increase data quality, we conducted a manual validation of the data. Four student assistants replaced erroneous answers and deleted nonsensical questions, question duplicates

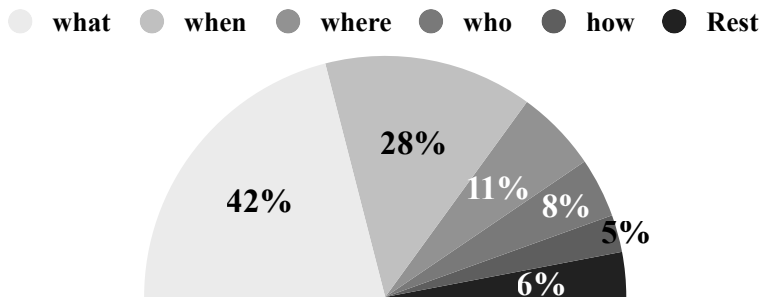



**Figure 10.3:** Distribution of question types

and incoherent texts.

During validation, 152 texts were found to be incoherent and discarded (along with all questions). Additionally, 3,388 questions were deleted because they were nonsensical or duplicates. 1,620 correct and 2,977 incorrect answers were exchanged, respectively, because they were inappropriate. If a question deletion resulted in texts without any questions, or if a text did not have any answerable questions, the text was also discarded.

After question validation, the final dataset comprises 9,935 questions that are labeled as script-based, 7,908 as text-based, and 1,978 as text-or-script.

### 10.2.2 Questions

Figure 10.3 gives the distribution over question types, which we extracted by looking at the first word in the question. The largest number of questions are *what* questions, most of which ask about participants of a script. *When* questions make up the second largest group, asking for temporal event structure. During the VP question experiment, some workers ignored the fact that we asked for temporal questions only, which resulted in a number of *how* questions. MCScript2.0 contains 50% questions labeled as script-based, which is a notably larger amount as compared to the approximately 27% of questions in MCScript labeled as script-based. The number of script-based questions varies between the question types, as can be seen in Figure 10.4. While *when* and *how* questions require script knowledge for finding an answer in more than 60% of cases, less than half of *what* questions do so. A simple explanation for this could be that *when* or *how* questions typically ask for events, while *what* questions ask for participants. Events are usually referred only once in a text, i.e. with the hiding of the respective event mention, the needed information has to be inferred. Participants in contrast tend to

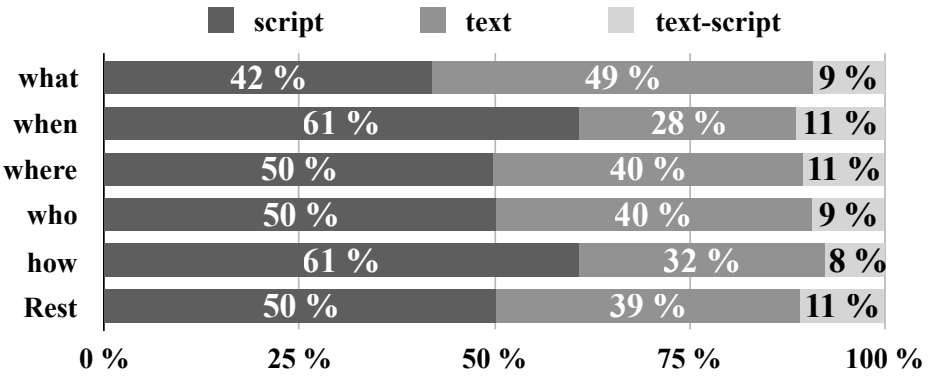



**Figure 10.4:** Proportion of labels per question type.

appear more often throughout a story.

Example 29 below illustrates this. Question 1 was originally asked about a sentence in which the plates are set for the dinner guests. The guests still appear in another sentence, so the answer can be inferred from the text. For question 2, in contrast, script knowledge is required for finding an answer: The event of *bringing the items to the table* is no longer mentioned, so the information that this happens typically after counting plates and silverware needs to be inferred.

(29) **T:** (...) I was told that there would be 13 or 14 guests. First I counted out 14 spoons, then the same number of salad forks, dinner forks, and knives. (...) I set each place with one napkin, one dinner fork, one salad fork, one spoon, and one knife. (...)
**Q1:** Who are the plate and cup for?
dinner guests ✓ the neighbor ✗
**Q2:** When did they bring the items over to the table?
after counting them ✓
after placing them on the table ✗

## 10.3 Experiments

We test three benchmark models on MCScript2.0 that were also evaluated on MCScript, so a direct comparison is possible. For the experiments, we split the data into a training set (14,191 questions on 2,500 texts), a development set (2,020 questions on 355 texts) and a test

set (3,610 questions on 632 texts). All texts of 5 randomly chosen scenarios were assigned completely to the test set, so a part of the test scenarios are unseen during training.

### 10.3.1 Models

#### Logistic Regression Classifier

As first model, we reimplemented the logistic regression classifier proposed by Merkhofer et al. (2018), which was also used in the adversarial filtering algorithm. The classifier employs 3 types of features: (1) Length features, encoding the length of the text, answer and questions on the word and character level, (2) overlap features, encoding the amount of literal overlap between text, question, and answers, and (3) binary lexical patterns encoding the presence or absence of words or combinations of words in answer, text and question.

#### Attentive Reader

As second model, we again employ an attentive reader (Hermann et al., 2015). As for MCScript, we adopt the formulation by Chen et al. (2016) (s. also Chapter 9.1). All tokens in text, question and answers are represented with word embeddings. Bi-directional gated recurrent units (GRUs, Cho et al. (2014)) process the text, question and answers and transform them into sequences of contextualized hidden states. The text is represented as a weighted average of the hidden states with a bilinear attention formulation, and another bilinear weight matrix is used to compute a scalar as score for each answer.

#### Three-way Attentive Network (TriAN)

As third model, we use a three-way attentive network (Wang et al., 2018), the best-scoring model of the shared task on MCScript[6]. Various types of information are employed to represent tokens: Word embeddings, part of speech tags, named entity embeddings, and word count/overlap features, similar to the logistic classifier. Three bidirectional LSTM (Hochreiter and Schmidhuber, 1997) modules are used to encode text, question and answers. The resulting hidden representations are reweighted with three attention matrices and then summed into vectors using three self-attention layers.

[6] Code available at https://github.com/intfloat/commonsense-rc

| | acc | $\text{acc}_{scr}$ | $\text{acc}_{txt}$ |
|---|---|---|---|
| *Logistic Model* | 0.61 | 0.56 | 0.67 |
| *Attentive Reader* | 0.65 | 0.63 | 0.68 |
| *TriAN* | **0.72** | **0.67** | **0.78** |
| *Humans* | 0.97 | | |

| | acc | $\text{acc}_{scr}$ | $\text{acc}_{txt}$ |
|---|---|---|---|
| *Logistic Model* | 0.79 | 0.76 | **0.81** |
| *Attentive Reader* | 0.72 | 0.75 | 0.71 |
| *TriAN* | **0.80** | **0.79** | **0.81** |
| *Humans* | 0.98 | | |

**Table 10.2:** Accuracy on test set, and on script/text-based questions ($\text{acc}_{scr}$, $\text{acc}_{txt}$) on **MCScript2.0** (left) and **MCScript** (right). The maximum per column is printed in **bold**.

Additionally, token representations are enhanced with *ConceptNet* (Speer et al., 2017) relations as a form of induced commonsense knowledge. ConceptNet is a large database of commonsense facts, represented as triples of two entities with a predicate. Relevant ConceptNet relations between words in the answer and the text are queried from the database and represented with relation embeddings, which are learned end-to-end during training and appended to the text token representations.

In contrast to Wang et al. (2018), we use the non-ensemble version of TriAN without pretraining on *RACE* (Lai et al., 2017), for better comparability to the other models.

### 10.3.2 Human Upper Bound

To assess human performance, 5 student assistants performed the reading comprehension task on 60 texts each. To assess agreement, 20 texts were annotated by all students. The annotators reached averaged pairwise agreement of 96.3% and an average accuracy of 97.4%, which shows that this is a simple task for humans.

### 10.3.3 Results

**Overall Performance.** The left hand side of Table 10.2 gives details on the performance of the three benchmark models on the test set, and on script-based ($\text{acc}_{scr}$) and text-based ($\text{acc}_{txt}$) questions in the test set. As can be seen, the logistic model scores worst, presumably because it has been used for the adversarial filtering algorithm and the data are thus

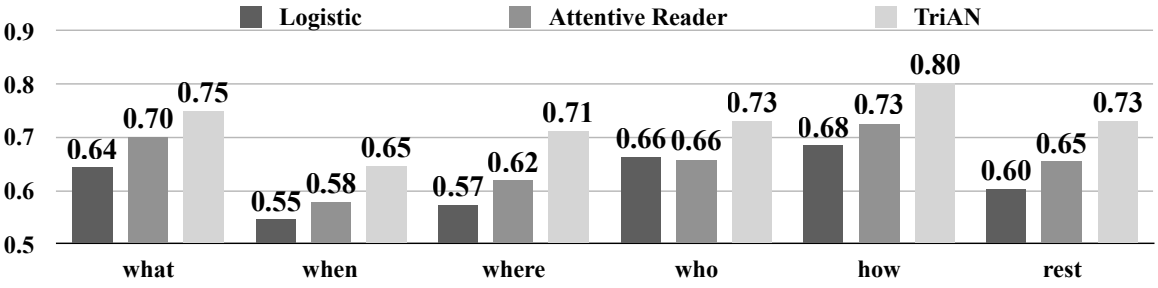



**Figure 10.5:** Performance of the models on question types.

most challenging for this model. TriAN performs best, clearly outperforming the attentive reader. TriAN is apparently superior in its way of text processing, since it employs a richer text representation and exploits attention mechanisms on more levels, which is reflected by a higher accuracy on text-based questions. In contrast, script-based questions seem to be challenging for TriAN. This is interesting, because it shows that ConceptNet alone cannot provide sufficient information for answering the kind of questions that can be found in MCScript2.0.

**Comparison to MCScript.** Since the same models were used for MCScript, a comparison of their performance on both datasets is possible. Results on MCScript are given on the right hand side of Table 10.2.[7] As can be seen, the performance of all three models is worse on MCScript2.0, showing that the dataset is generally more challenging. In contrast to MCScript, script-based questions in MCScript2.0 are clearly harder to answer than text-based questions: All models perform worse on script-based questions compared to text-based questions. In comparison to MCScript, the performance of TriAN is 12% lower. This indicates that the new mode of question collection and the answer selection via adversarial filtering resolve some of the difficulties with MCScript.

To assess whether the performance difference to MCScript is due to the 90 new scenarios being more challenging, we additionally evaluated the models on these scenarios only. We found no performance difference on the new vs. old scenarios.

**Influence of Adversarial Filtering.** To find out how large the influence of the new question collection method and the answer selection via adversarial filtering is, we conducted an additional experiment: We applied the original answer selection method of MCScript

[7] All models except the attentive reader were retrained (and in the case of the logistic model re-implemented), since no exact numbers on script/text-based questions were published.

to MCScript2.0 to create an alternative version of the data that is not based on adversarial filtering. Correct answers were selected to have the lowest possible overlap with the reading text. Incorrect answers were selected using the majority voting technique described in Section 10.1.4.

We found that the adversarial filtering accounts for around two thirds of the total accuracy difference of TriAN as compared to MCScript, i.e. one third of the difference can be attributed to the new question collection. This means that both modifications together add to the larger difficulty of MCScript2.0.

**Question Types.** Figure 10.5 shows the performance of the models on single question types, as identified in Section 4. It is clear that *when* questions are most challenging for all models. The logistic classifier performs almost at chance level. As far as TriAN is concerned, we found that many cases of errors ask for the typical temporal order of events, as Example 30 illustrates:

(30) **Q:** When did they put the nozzle in their tank?
before filling up with gas. ✓
after filling up with gas. ✗

The event of *put the nozzle in the tank* is not mentioned in the shown version of the text, so it is not possible to read off the text when the event actually took place.

*How* questions are the least difficult questions. This can be explained with the fact that many *how* questions ask for numbers that are mentioned in the text (e.g. *How long did they stay in the sauna?* or *How many slices did they place onto the paper plate?*). The answer to such questions can often be found with a simple text lookup. Another part of how questions asks for the typical duration of an activity. These questions often have similar answers irrespective of the scenario, since most of the narrations in MCScript2.0 span a rather short time period. Such answers can easily be memorized by the models.

Especially for TriAN, *what* and *who* questions seem to be easy. This could be explained with the fact that ConceptNet contains lots of information about entities and their relations to each other, apparently also covering some information about script participants, which seems to be useful for these question types.

## 10.4 Conclusion

In this chapter, we presented MCScript2.0, a new machine comprehension dataset with a focus on challenging inference questions that require script knowledge or commonsense knowledge for finding the correct answer. Our new question collection procedure, which is focused on questions, the answer to which is hidden in the text and thus needs to be inferred from background knowledge, results in about half of the questions in MCScript2.0 requiring such inference, which is a much larger amount compared to MCScript.

By altering the question collection procedure and combining this with an adversarial filtering algorithm for collecting challenging answer candidates, we create questions that eliminate two undesired features of MCScript: First, questions are based on specific texts rather than a general scenario, which means that they are less repetitive within texts of the same scenario. Second, our focus on noun and verb phrase within the text naturally result on a larger focus on participants and events in the text, narrowing the focus of our evaluation paradigm on script knowledge.

We evaluate several benchmark models on MCScript2.0 and show that even a state-of-the-art model, which makes use of ConceptNet as a source for commonsense knowledge, but does not utilize script knowledge, struggles to answer many question in our corpus. We find that the script-based questions in MCScript2.0 are harder to answer than text-based questions, which indicates that a more complex inference over background knowledge is required for finding an answer for these questions. In particular, we expect that models which utilize script knowledge will perform superior on the data, since our data are focused on script events and participants.

# Part IV

# Future Work and Conclusion

Chapter 11

# Outlook

This dissertation provides important building blocks in the field of script parsing and applications of script knowledge. However, there are many open problems and tasks that require further research. In this chapter, we give a brief overview of promising future directions for applications and models of script knowledge in natural language understanding. [1]

**Script Parsing.** In this dissertation, we focused only on some of the most central subtasks of script parsing. For example, we only concentrated on script parsing based on events. However, for a full-scale script parsing model, it is necessary to also address participant labeling. A participant labeling model should predict the participant types of noun phrases in a text based on an underlying participant representation. We also restricted ourselves to a single scenario per text. Usually in naturally occurring texts, multiple script scenarios are addressed (cf. the examples from the Spinn3r data (Burton et al., 2009), e.g. in Chapter 2). This means that a full-scale parsing model also needs to address several different script scenarios per text. A first step towards this direction has been done by Wanzare et al. (2019), who propose an approach to segment texts into parts that talk about a specific scenario. Each text segment is then assigned a script scenario label. A first step for script parsing with multiple scripts is to apply our script parser on segments identified by Wanzare's model,

[1] Parts of the following sections follow a confidential proposal for a continuation of the project in the context of which this dissertation was written. The author of this dissertation has contributed to this proposal.

after training the parser on ESDs of the respective scenario.

*Methods.* We think it is important to not just look at such tasks in isolation, but to implement integrated approaches, where information from one subtask can be used for other subtasks. Neural networks excel at such integration of several subtasks, so an important direction for future research is the application of deep learning techniques to script parsing: A neural network can for example be trained to simultaneously predict event and participant labels. Script parsing based on deep learning was not in the focus of this thesis. One of the main reasons for this is that available script resources are too small to train deep models with many parameters. There are several options to generate more training data, one being *bootstrapping*. The idea is to start out with a small, but clean crowdsourced script database. In each iteration of the bootstrapping algorithm, a number of texts is parsed. The parsed events are then added to the training data for the next iteration.

**Non-Discrete Event Representations.** One central advantage of using a neural network for script parsing is that it is possible for such a network to learn interesting new event representations that are continuous rather than discrete. So-called *event embeddings* form an important alternative script representation, as opposed to discrete labeled paraphrase sets[2]. The idea is related to Modi and Titov (2014), who learn event embeddings for a temporal ordering model. Event embeddings are similar to word embeddings based on pretrained language models, which have shown great success in a wide range of NLP tasks recently (Peters et al., 2018; Devlin et al., 2019). It can thus be assumed that pretrained event embeddings can also successfully be utilized for downstream tasks.

Pretrained language models are trained to predict missing words. Correspondingly, event embeddings could be computed as part of an event prediction model. The idea is to train a network to predict missing events based on a chain of given events, i.e. a narrative cloze task. Events are composed out of their verbs and arguments, and represented as continuous vectors. Similar to word embeddings, the resulting event embeddings can then be used to augment sentence representations for various downstream tasks, under the assumption that they encode information about the surrounding script structure.

[2] Since a neural script parser would learn such an internal event representation, it can thus also be seen as a model for script induction.

A similar idea was proposed by Hong et al. (2018), who learned event embeddings in a multi-task setting, based on a role filler task, and utilized them for SRL and participant prediction. They mostly assume an event representation that is close to frame semantics, but learning continuous script event representations works in a similar fashion.

**Generative Script Models.** There are other kinds of script representations beyond paraphrase sets and event embeddings that can be explored, such as generative latent variable models. One option for learning such a latent script model is Bayesian learning. In such an approach, events are not represented as clusters, but as latent variables that are used to *generate* text. The generation step of the Bayesian model can be seen as an inverse parsing step: While in script parsing, an event type is assigned based on a text snippet, here, a text that describes the latent event is generated from the latent event representation.

Similarly, neural latent variable models have been shown to be effective for a range of tasks, including the script domain: Weber et al. (2018) use variational autoencoders (Kingma and Welling, 2013) for a narrative cloze model based on news texts. They argue that the learned latent variables represent different states of a news script. Applied to a script parsing task with everyday narrations, the latent variables would no longer necessarily correspond to event types, but rather represent a script in terms of the current "state" of the story and its participants.

**Semantic Role Labeling and Implicit Argument Prediction.** Frame semantics and script knowledge are highly interconnected. We thus believe that script knowledge helps in the area of semantic role labeling (SRL). Roth and Lapata (2015) for example have successfully used discourse features for improving a SRL model. Since script knowledge contains also discourse-level information, this indicates that script knowledge is important for SRL.

Another interesting related task is implicit argument induction, which could benefit from the use of script knowledge. The task is to predict non-realized verbal arguments. For this, script knowledge is highly relevant, for example in a case in which a participant is not realized explicitly. The simplest way to approach the problem would be to use a script parser on a text and feed the event information into an implicit argument prediction model. Modi et al. (2017) have looked at a similar problem, namely discourse referent prediction, i.e. the

prediction of explicitly realized upcoming referents. They use a range of script-based features, which can also be utilized to predict implicit arguments.

**Modeling Temporal Event Order in Machine Comprehension.** As addressed in this thesis, machine comprehension is an application that could benefit from the use of script knowledge. For example, we found that existing models struggle with questions that ask for the typical temporal ordering of events.

Script knowledge could be used to improve machine comprehension models, such that they are capable of modeling temporal order. As mentioned earlier, the event embeddings computed by Modi and Titov (2014) encode temporal order, so they could be utilized for a machine comprehension model.

Modi's model outputs scores for each event that could be used directly, as preliminary experiments have indicated. Example 31 shows the output of a simple re-implementation of Modi's model that was trained on ESDs for a total of over 200 scenarios. The model was used to score the events in question and both answers. As can be seen, the model correctly orders the events of *crack eggs*, *put into pan* and *cook*, as is indicated by the descending scores.

(31) Q. When did they crack 4 eggs one by one? *5.6*

a. Before putting them in the pan *5.4*

b. After cooking in the pan *5.2*

There are at least two options to use such a model. First, it can be part of an ensemble: One component of the ensemble model would be the vanilla machine comprehension model, and the other component could use the predicted ordering scores exclusively to make a prediction[3]. Second, the learned event embeddings can also be used directly, since they encode ordering information. For example, they can be appended or combined with the question and answer representations of a machine comprehension model. This will leverage the inherent ordering information for the answer prediction.

[3] Of course, the model should know how to handle *before* and *after*, which should be taken into account for making the prediction.

**Scripts in Real World Applications.** This dissertation looked at scripts from a more theoretical perspective. In the future, it will be desirable to carry these theoretical considerations into actual applications and technologies in the area of NLU that are used daily by humans, such as more intelligent personal assistants. Script knowledge is for example relevant for some aspects of information retrieval and question answering as conducted by programs as *Google Assistant*. The work on machine comprehension with scripts conducted in this dissertation is a first step towards this direction. Another example would be a personal assistant with access to the various scripts involved when a user is taking a vacation: It first directs the user in getting a flight, then recommends restaurants at the goal destination, helps the user with the check-in at the hotel, and guides her to the nearest beach afterwards.

**Script Knowledge and Pretrained Transformer Models.** Recent developments in computational linguistics and natural language processing have shown that large, pretrained transformer language models such as ELMo (Peters et al., 2018), BERT (Devlin et al., 2019) or XLNet (Yang et al., 2019) perform extremely successful in a wide range of tasks, covering various question answering tasks, natural language inference, and others. Due to their outstanding performance, such models replace many technologies that have been developed over decades. This especially also affects knowledge-based systems that use hard-coded resources, which has been a standard for a long time. In our work, we use script knowledge in terms of a symbolic knowledge base. Thus, the question arises to what extent phenomena that require a model to use script knowledge are already covered by pretrained transformers. It can be expected that such models perform well even on challenging commonsense inference data, but we expect that a subset of cases cannot be modeled without taking explicit script knowledge into consideration. This is due to the fact that many aspects of scripts are not realized in texts and thus cannot be learned by a model that is purely trained on texts.

Chapter 12

# Conclusion

The field of natural language understanding includes a range of tasks and approaches around the automated comprehension of text. Script knowledge is an important component for NLU systems. In human text comprehension, scripts are common ground, and thus they are rarely made explicit in texts. This makes it hard for NLU systems to leverage such knowledge. This dissertation project addressed some central questions about how script knowledge can be used for NLU. The main contributions of this work are the following:

1. Script parsing is an important prerequisite for leveraging script knowledge in NLU applications. The aim of script parsing is to detect the events and participants of a script that are mentioned in a text. This allows a model to anchor the text with the script and to then leverage script information that is unmentioned. *We have established the first script parsing model* that can be scaled up to large script collections. It is implemented as a conditional random field that is trained on ESDs and that makes essential use of the inherent ordering information of such event sequences. *Thus, we have shown that ordering information plays an important role for script parsing.*

2. *We have provided the first data set for the systematic evaluation of script parsing models. InScript* is a corpus of narrative texts that is fully annotated with script event types and script participant types. We also provided a thorough analysis of the linguistic phenomena that need to be modeled when applying a script parsing model to our texts.

*Our analyses and annotations show the importance of representing events in terms of large paraphrase sets.* We show that paraphrase sets massively reduce the difficulty of automatic text-to-script mapping and increase its accuracy.

3. *We have established an end-to-end task in NLU to asses the contribution of script knowledge for machine comprehension.* To this end, we have created MCScript, a dataset of narrative texts and questions on the texts. By collecting the questions based on a scenario rather than a text, we aimed at creating challenging inference questions, which require a system to use script knowledge for finding an answer. We conducted a shared task on the data, which has shown that script knowledge is not required to perform well. Analyses showed that this could be attributed to the mode of question collection. We consequentially revised the question collection and created a second data set, MCScript2.0. First experiments with state-of-the art models show that MCScript2.0 is much more challenging than MCScript and that models that don't use script knowledge perform poorly.

This work provides the first steps for making script knowledge accessible for NLU. Yet, many complex problems are left open that need to be solved.

To this end, we pointed out possible directions for future work, ranging from new script parsing methods to diverse applications of scripts in the field of computational linguistics/NLU and in the real world. We feel that although we restricted ourselves to central questions in the area of script-based NLU, we achieved some substantial results and build a solid foundation for future work. We hope that our research efforts will raise important awareness for the question on how script knowledge can be utilized in NLU in general, and that it highlights the possible contribution of script knowledge for intelligent text processing systems.

# List of Figures


1 Ein Beispiel-Graph für das *ein Restaurant besuchen* - Skript. . . . . vi

2 Ein Beispiel für Ereignis-basiertes Skript-Parsing. . . . . vii

3 Ein Beispiel für einen kurzen Text mit zwei Leseverstehensfragen. . . . . ix

1.1 A failed search query in WolframAlpha. . . . . 4

1.2 An example script graph for the GOING TO A RESTAURANT scenario. . . . . 6

1.3 Representing events as paraphrase sets. . . . . 7

1.4 The *Commerce_Pay* frame, from FrameNet (Ruppenhofer et al., 2006) . . . . 8

1.5 Frame analysis (left) and Script analysis (right). . . . . 9

1.6 An example for script parsing. . . . . 11

1.7 The narrative order of events can deviate from the order in which they actually took place. . . . . 12

1.8 The granularity of event mentions can differ a lot. . . . . 12

1.9 Two example stories form the ROC stories dataset (Mostafaza deh et al., 2016). 13

1.10 An example story from the Spinn3r data set (Burton et al., 2009) that talks about the HAVING DINNER scenario (in blue), as well as BAKING A CAKE (in green). . . . . 14

1.11 Script knowledge can improve coreference resolution systems. . . . . 17

1.12 An example generated story from Shkadzko (2017). . . . . 18

2.1 Representing scripts as paraphrase sets, based on crowdsourced event sequence descriptions. . . . . 27
2.2 Representing scripts as narrative chains, based on text collections. . . . . 29
2.3 A small excerpt of *ConceptNet*, taken from Speer and Havasi (2012). . . . . 35

3.1 An example of script parsing with a story fragment and an excerpt of the BAKING A CAKE script. . . . . 39
3.2 Text Segmentation and Scenario identification based on a script database. 40
3.3 Event Verb Identification, Event Type Identification and Participant Identification. . . . . 41
3.4 Example text from the Spinn3r blog stories corpus (Burton et al., 2009). Script events are underlined, script-evoking events are marked in *italics*. . . . . 43

4.1 An example of script parsing with an excerpt of the BAKING A CAKE script and a story snippet. . . . . 49
4.2 BathAnnotation . . . . . 54
4.3 Inter-annotator agreement statistics. . . . . 62
4.4 The number of participants and events in the templates. . . . . 63
4.5 Annotation statistics over all scenarios. . . . . 64
4.6 The percentage of stories in the BAKING A CAKE scenario that contain a certain participant label. . . . . 64
4.7 Distribution of participants (left) and events (right) for the first, the top 2 to 5 and the top 6 to 10 most frequently appearing events/participants, SCREV/SCRPART_OTHER and the rest. . . . . 65
4.8 Average number of participant mentions for a story, for the first, the top 2 to 5 and the top 6 to 10 most frequently appearing events/participants, and the overall average. . . . . 65
4.9 MTLD values for DeScript and InScript, per scenario. . . . . 66
4.10 Entropy over verb lemmas for events (left y-axis, $H(x)$) in the GOING ON A TRAIN SCENARIO. Bars in the background indicate the absolute number of occurrence of instances (right y-axis, $N(x)$). . . . . 67

5.1 Aligning two text snippets of varying difficulty with the BAKING A CAKE script. 69
5.2 An example annotation in InScript (lower part), and the corresponding paraphrase sets from DeScript (upper part), for the PLANTING A TREE scenario. . 71
5.3 A full script with participant paraphrase sets. . . . . 72
5.4 Combining the EDs in the DIG event into patterns (upper part) and the actual verb annotation (lower part). . . . . 73
5.5 Aligning participants with a pattern (left), selecting a noun from the participant paraphrase set and labeling (right). . . . . 75
5.6 Entailment types. . . . . 77
5.7 Combination table for lexical entailment classes. . . . . 78
5.8 One fully labeled event instance, with compositionally derived clausal entailment label (*Reverse Entailment*). . . . . 78
5.9 Distribution of composed labels. . . . . 80

6.1 Structure of the proposed parser. . . . . 83
6.2 An event type identification model based on textual similarity. . . . . 87
6.3 A CRF-based script parser. . . . . 88
6.4 Accuracy of the CRF model on clausal entailment classes. . . . . 97
6.5 Accuracy of the baseline models on the entailment classes: $\text{sim}_{w2v}$ (left) and $\text{sim}_{lemma}$ (right). . . . . 98

7.1 Example text fragment from NewsQA . . . . . 109
7.2 Example text fragment from ReCoRD . . . . . 109
7.3 Example text fragment from TriviaQA . . . . . 110
7.4 Example text fragment from TriviaQA . . . . . 115
7.5 Example item from the Winograd schema challenge . . . . . 116

8.1 An example for a text snippet with two reading comprehension questions. . 119
8.2 Distribution of question types in the data. . . . . 125
8.3 An example text with 2 questions from MCScript . . . . . 126

9.1 Accuracy values of the baseline models on question types appearing $> 25$ times. . . . . 135

9.2 The importance of features, with the top 10 lexicalized patterns on the left and overlap/length features on the right. $w$ is the actual feature weight. The product of the weights magnitude with the standard deviation ($\sigma|w|$) balances rare and more common features and is used for ranking the features. Both tables taken from Merkhofer et al. (2018). . . . . 146

10.1 Example text fragment from MCScript2.0 . . . . . 153

10.2 Screenshot of an item in the participant question collection experiment. . . 156

10.3 Distribution of question types . . . . . 161

10.4 Proportion of labels per question type. . . . . 162

10.5 Performance of the models on question types. . . . . 165

1 Annotation view for an instance of the event preheat oven from the cake-scenario. The sentence has to be aligned to line 05. . . . . 225

# List of Tables


2.1 Overview of models of script knowledge. The evaluation tasks are narrative cloze/event prediction (NC), event paraphrasing (P) and event ordering (O). 26

4.1 Corpus statistics for different scenarios (standard deviation given in parentheses). The maximum per column is highlighted in **boldface**, the minimum in ***boldface italics***. . . . . 52

4.2 Bath scenario template (labels added in the second phase of annotation are marked in bold). . . . . 55

5.1 Distribution of lexical labels on events and participants. . . . . 79

6.1 Identification of script-relevant verbs within a scenario and independent of the scenario. The maximum per column is printed in **bold**. . . . . 92

6.2 Event Type Classification performance, for the similarity models ($sim_{lemma}$, based on lemma similarity, and $sim_{w2v}$, based on word2vec); and for the CRF model, with and without sequential features. The maximum per column is printed in **bold**. . . . . 93

6.3 The influence of the sequential feature per scenario. . . . . 94

6.4 Full text-to-script mapping results. The maximum per column is printed in **bold**. . . . . 94

8.1 Distribution of question categories . . . . . 125

9.1 Accuracy of the baseline systems on text-based (*Text*), on commonsense-based questions (*CS*), and on the whole test set (*Total*). All numbers are percentages. 134

9.2 Overview of techniques and resourced used by the participating systems. . 137

9.3 The accuracy of participating systems and the two baselines in total, on commonsense-based questions, text-based questions and on out-of-domain questions (from the 5 held-out testing scenarios). The best performance for each column is marked in **bold print**. Significant differences in results between two adjacent lines are marked by an asterisk (* p<0.05) in the upper line. The last line shows the human upper bound on MCScript as comparison. 140

9.4 Accuracy of participating systems and the baselines on the six most frequent question types. The best performance for each column is marked in **bold print**. Significant differences in results between two adjacent lines are marked by an asterisk (* p<0.05) in the upper line. . . . . 141

10.1 Distribution of question labels, before validation. . . . . 160

10.2 Accuracy on test set, and on script/text-based questions ($acc_{scr}$, $acc_{txt}$) on **MCScript2.0** (left) and **MCScript** (right). The maximum per column is printed in **bold**. . . . . 164

1 Event Annotation. . . . . 204

2 Participant Annotation. . . . . 207

# Bibliography


Lea T. Adams and Patricia E. Worden. Script development and memory organization in preschool and elementary school children. Discourse Processes, 9(2):149–166, 1986.

Frederik Arnold. Extending Script-Knowledge with Pre-Conditions. Master's thesis, Universität des Saarlandes, 2016.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. The Semantic Web, pages 722–735, 2007.

Amit Bagga and Breck Baldwin. Algorithms for Scoring Coreference Chains. In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference, pages 563–566, 1998.

Avron Barr and Edward A. Feigenbaum. The Handbook of Artificial Intelligence. Addison-Wesley, 1981.

Cosmin Bejan and Sanda Harabagiu. Unsupervised Event Coreference Resolution with Rich Linguistic Features. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1412–1422, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Francis Bond and Ryan Foster. Linking and Extending an Open Multilingual Wordnet. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics

(Volume 1: Long Papers), pages 1352–1362, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Kevin Burton, Akshay Java, and Ian Soboroff. The ICWSM 2009 Spinn3r Dataset. In Third Annual Conference on Weblogs and Social Media (ICWSM 2009), San Jose, CA, May 2009. AAAI.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Toward an Architecture for Never-ending Language Learning. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10, pages 1306–1313, Atlanta, Georgia, 2010. AAAI Press.

Nathanael Chambers. Behind the scenes of an evolving event cloze test. In Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics, pages 41–45, Valencia, Spain, April 2017. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. Unsupervised Learning of Narrative Event Chains. In Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 789–797, Columbus, Ohio, June 2008. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. Unsupervised Learning of Narrative Schemas and their Participants. In 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, pages 602–610, Singapore, August 2009. Association for Computational Linguistics.

Danqi Chen, Jason Bolton, and Christopher D. Manning. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. In Proceedings of the 54th Annual

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2358–2367, Berlin, Germany, August 2016. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

Richard Edward Cullingford. Script application: Computer understanding of newspaper stories. Technical report, DTIC Document, 1978.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL Recognising Textual Entailment Challenge. In Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment, MLCW'05, pages 177–190, Southampton, UK, 2006. Springer-Verlag.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Gated-Attention Readers for Text Comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1832–1846, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Peng Ding and Xiaobing Zhou. YNU_Deep at SemEval-2018 Task 11: An Ensemble of Attention-based BiLSTM Models for Machine Comprehension. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1043–1047, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Pedro Domingos. The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. Basic Books, 2015.

Michael George Dyer. In-Depth Understanding. A Computer Model of Integrated Processing for Narrative Comprehension. Technical report, DTIC Document, 1982.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. The Journal of Machine Learning Research, 9:1871–1874, June 2008.

Christiane Fellbaum. WordNet: An Electronic Lexical Database. MIT Press, 1998.

Charles J. Fillmore. Frame semantics, pages 111–137. Hanshin Publishing Co., Seoul, South Korea, 1982.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. Psychological bulletin, 76(5):378–382, 1971.

Eibe Frank, Mark A. Hall, and Ian H. Witten. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". 2016.

Lea Frermann, Ivan Titov, and Manfred Pinkal. A Hierarchical Bayesian Model for Unsupervised Induction of Script Knowledge. In Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 49–57, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

Annemarie Friedrich, Alexis Palmer, and Manfred Pinkal. Situation entity types: automatic classification of clause-level aspect. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1757–1768, Berlin, Germany, August 2016. Association for Computational Linguistics.

William A. Gale, Kenneth W. Church, and David Yarowsky. One Sense Per Discourse. In Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992, 1992.

Daniel Gildea and Daniel Jurafsky. Automatic Labeling of Semantic Roles. Computational Linguistics, 28(3):245–288, 2002.

José-Ángel González, Lluís-F. Hurtado, Encarna Segarra, and Ferran Pla. ELiRF-UPV at SemEval-2018 Task 11: Machine Comprehension using Commonsense Knowledge. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1034–1037, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Jonathan Gordon and Benjamin Van Durme. Reporting Bias and Knowledge Acquisition. In Proceedings of the 2013 Workshop on Automated Knowledge Base Construction, AKBC '13, pages 25–30, San Francisco, California, USA, 2013. ACM.

Arthur C. Graesser, Sallie E. Gordon, and John D. Sawyer. Recognition memory for typical and atypical actions in scripted activities: Tests of a script pointer + tag hypothesis. Journal of Verbal Learning and Verbal Behavior, 18(3):319 – 332, 1979. ISSN 0022-5371.

David Graff. English Gigaword. Linguistic Data Consortium, 2002.

Mark Granroth-Wilding and Stephen Clark. What Happens Next? Event Prediction Using a Compositional Neural Network Model. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, pages 2727–2733, Phoenix, Arizona, 2016. AAAI Press.

H Paul Grice. Logic and conversation. 1975, pages 41–58, 1975.

Timo Gühring, Nicklas Linz, Rafael Theis, and Annemarie Friedrich. SWAN: an easy-to-use web-based annotation system. In Proceedings of the 13th Conference on Natural Language Processing (KONVENS). September 19-22, Bochum, Germany, 2016.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, pages 1693–1701, Montreal, Canada, 2015. MIT Press.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations. arXiv preprint arXiv:1511.02301, 2015.

L. Hirschman and R. Gaizauskas. Natural Language Question Answering: The View from Here. Natural Language Engineering, 7(4):275–300, December 2001. ISSN 1351-3249.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. Neural computation, 9:1735–80, 12 1997.

Xudong Hong, Asad Sayeed, and Vera Demberg. Learning distributed event representations with a multi-task approach. In Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, pages 11–21, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Bram Jans, Steven Bethard, Ivan Vulić, and Marie Francine Moens. Skip N-grams and Ranking Functions for Predicting Script Events. In Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 336–344, Avignon, France, April 2012. Association for Computational Linguistics.

Zhengping Jiang and Qi Sun. CSReader at SemEval-2018 Task 11: Multiple Choice Question Answering as Textual Entailment. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1053–1057, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Alexander Kampmann, Stefan Thater, and Manfred Pinkal. A case-study of automatic participant labeling. In Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology (GSCL 2015), pages 97–105, 2015.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking Beyond the Surface: A Challenge Set for Reading Comprehension over Multiple Sentences. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 252–262, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2013.

Dan Klein and Christopher D. Manning. Fast Exact Inference with a Factored Model for Natural Language Parsing. In S. Becker, S. Thrun, and K. Obermayer, editors, Advances in Neural Information Processing Systems 15, pages 3–10. MIT Press, 2003.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding Comprehension Dataset From Examinations. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

J. Richard Landis and Gary G. Koch. The Measurement of Observer Agreement for Categorical Data. Biometrics, 33(1):pp. 159–174, 1977.

Heeyoung Lee, Marta Recasens, Angel Chang, Mihai Surdeanu, and Dan Jurafsky. Joint Entity and Event Coreference Resolution across Documents. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 489–500, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web, 6(2):167–195, 2015.

Douglas B. Lenat. CYC: A Large-scale Investment in Knowledge Infrastructure. Communications of the ACM, 38(11):33–38, November 1995. ISSN 0001-0782.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet Physics Doklady, volume 10, pages 707–710, 1966.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The Winograd Schema Challenge. In Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, KR'12, pages 552–561, Rome, Italy, 2012. AAAI Press.

Yongbin Li and Xiaobing Zhou. ZMU at SemEval-2018 Task 11: Machine Comprehension Task using Deep Learning Models. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1073–1077, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Dekang Lin. An information-theoretic definition of similarity. In Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

Xiao Ling and Daniel S. Weld. Temporal Information Extraction. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10, pages 1385–1390, Atlanta, Georgia, 2010. AAAI Press.

Qingxun Liu, Hongdou Yao, Xaobing Zhou, and Ge Xie. YNU_AI1799 at SemEval-2018 Task 11: Machine Comprehension using Commonsense Knowledge of Different model ensemble. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1038–1042, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Bill MacCartney and Christopher D. Manning. Natural Logic for Textual Inference. In Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing, pages 193–200, Prague, June 2007. Association for Computational Linguistics.

Bill MacCartney and Christopher D. Manning. An extended model of natural logic. In Proceedings of the Eight International Conference on Computational Semantics, pages 140–156, Tilburg, The Netherlands, January 2009. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics:

System Demonstrations, pages 55–60, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in Translation: Contextualized Word Vectors. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pages 6297–6308, Long Beach, California, USA, 2017. Curran Associates Inc.

Philip M. McCarthy. An Assessment of the Range and Usefulness of Lexical Diversity Measures and the Potential of the Measure of Textual, Lexical Diversity (MTLD). PhD thesis, The University of Memphis, 2005.

Philip M. McCarthy and Scott Jarvis. MTLD, vocd-D, and HD-D: A validation study of sophisticated approaches to lexical diversity assessment. Behavior Research Methods, 2010.

Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al. Personalized speech recognition on mobile devices. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5955–5959. IEEE, 2016.

Elizabeth Merkhofer, John Henderson, David Bloom, Laura Strickhart, and Guido Zarrella. MITRE at SemEval-2018 Task 11: Commonsense Reasoning without Commonsense Knowledge. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1078–1082, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Risto Miikkulainen. Subsymbolic natural language processing: An integrated model of scripts, lexicon, and memory. MIT press, 1993.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges,

L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 3111–3119. Curran Associates, Inc., 2013b.

George A. Miller. WordNet: A Lexical Database for English. Commununications of the ACM, 38(11):39–41, November 1995.

Anne-Lyse Minard, Manuela Speranza, Eneko Agirre, Itziar Aldabe, Marieke van Erp, Bernardo Magnini, German Rigau, and Rubén Urizar. SemEval-2015 Task 4: TimeLine: Cross-Document Event Ordering. In Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), pages 778–786, Denver, Colorado, June 2015. Association for Computational Linguistics.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending Learning. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15, pages 2302–2310, Austin, Texas, 2015. AAAI Press.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending Learning. Communications of the ACM, 61(5):103–115, April 2018.

Ashutosh Modi. Event Embeddings for Semantic Script Modeling. In Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, pages 75–83, Berlin, Germany, August 2016. Association for Computational Linguistics.

Ashutosh Modi and Ivan Titov. Inducing Neural Models of Script Knowledge. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning, pages 49–57, Ann Arbor, Michigan, June 2014. Association for Computational Linguistics.

Ashutosh Modi, Tatjana Anikina, Simon Ostermann, and Manfred Pinkal. InScript: Narrative texts annotated with script information. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), pages 3485–3493, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

Ashutosh Modi, Ivan Titov, Vera Demberg, Asad Sayeed, and Manfred Pinkal. Modeling Semantic Expectation: Using Script Knowledge for Referent Prediction. Transactions of the Association for Computational Linguistics, 5:31–44, 2017.

Gordon E. Moore. Cramming more components onto integrated circuits. Electronics, 38.8: 114–117, 1965.

Nasrin Mostafaza deh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics.

Erik T. Mueller. Understanding script-based stories using commonsense reasoning. Cognitive Systems Research, 5(4):307–340, 2004.

Erik T. Mueller. Story understanding. Encyclopedia of Cognitive Science, 2006.

Simon Ostermann, Michael Roth, Stefan Thater, and Manfred Pinkal. Aligning Script Events with Narrative Texts. In Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), pages 128–134, Vancouver, Canada, August 2017. Association for Computational Linguistics.

Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. MCScript: A Novel Dataset for Assessing Machine Comprehension Using Script Knowledge. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018), Miyazaki, Japan, May 2018a. European Languages Resources Association (ELRA).

Simon Ostermann, Michael Roth, Ashutosh Modi, Stefan Thater, and Manfred Pinkal. SemEval-2018 Task 11: Machine Comprehension Using Commonsense Knowledge. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 747–757, New Orleans, Louisiana, June 2018b. Association for Computational Linguistics.

Simon Ostermann, Hannah Seitz, Stefan Thater, and Manfred Pinkal. Mapping Texts to Scripts: An Entailment Study. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018), Miyazaki, Japan, May 2018c. European Languages Resources Association (ELRA).

Simon Ostermann, Michael Roth, and Manfred Pinkal. MCScript2.0: A Machine Comprehension Corpus Focused on Script Events and Participants. In Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019), pages 103–117, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Alexis Palmer and Annemarie Friedrich. Genre distinctions and discourse modes: Text types differ in their situation type distributions. In Proceedings of the Workshop on Frontiers and Connections between Argumentation Theory and Natural Language Processing, Forlì-Cesena, Italy, 2014.

Martha Palmer, Daniel Gildea, and Nianwen Xue. Semantic role labeling. Synthesis Lectures on Human Language Technologies, 3(1):1–103, 2010.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Karl Pichotta and Raymond Mooney. Statistical Script Learning with Multi-Argument Events. In Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 220–229, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

Karl Pichotta and Raymond J. Mooney. Learning statistical scripts with LSTM recurrent neural networks. In Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16), Phoenix, Arizona, 2016.

Martin F Porter. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.

Susanne Raisig, Tinka Welke, Herbert Hagendorf, and Elke Van Der Meer. Insights Into Knowledge Representation: The Influence of Amodal and Perceptual Variables on Event Knowledge Retrieval From Memory. Cognitive Science, 33(7):1252–1266, 2009.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

Michaela Regneri. Event Structures in Knowledge, Pictures and Text. PhD thesis, Universität des Saarlandes, 2013.

Michaela Regneri and Rui Wang. Using Discourse Information for Paraphrase Extraction. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 916–927, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

Michaela Regneri, Alexander Koller, and Manfred Pinkal. Learning Script Knowledge with Web Experiments. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 979–988, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Michaela Regneri, Alexander Koller, Josef Ruppenhofer, and Manfred Pinkal. Learning Script Participants from Unlabeled Data. In Proceedings of the International Conference

Recent Advances in Natural Language Processing 2011, pages 463–470, Hissar, Bulgaria, September 2011. Association for Computational Linguistics.

Philip Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'95, pages 448–453, Montreal, Quebec, Canada, 1995. Morgan Kaufmann Publishers Inc.

Sofia Reznikova and Leon Derczynski. IUCM at SemEval-2018 Task 11: Similar-Topic Texts as a Comprehension Knowledge Source. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1068–1072, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 193–203, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

Michael Roth and Mirella Lapata. Context-aware Frame-Semantic Role Labeling. Transactions of the Association for Computational Linguistics, 3:449–460, 2015.

Michael Roth and Mirella Lapata. Neural Semantic Role Labeling with Dependency Path Embeddings. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1192–1202, Berlin, Germany, August 2016. Association for Computational Linguistics.

Rachel Rudinger, Vera Demberg, Ashutosh Modi, Benjamin Van Durme, and Manfred Pinkal. Learning to predict script events from domain-specific text. In Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics, pages 205–210, Denver, Colorado, June 2015a. Association for Computational Linguistics.

Rachel Rudinger, Pushpendre Rastogi, Francis Ferraro, and Benjamin Van Durme. Script Induction as Language Modeling. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1681–1686, Lisbon, Portugal, September 2015b. Association for Computational Linguistics.

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. FrameNet II: Extended Theory and Practice. 2006.

Roger C. Schank and Robert P. Abelson. Scripts, Plans, and Knowledge. In Proceedings of the 4th international joint conference on Artificial intelligence-Volume 1, pages 151–157. Morgan Kaufmann Publishers Inc., 1975.

Claude E. Shannon. A Mathematical Theory of Communication. The Bell System Technical Journal, 27(3):379–423, 1948.

Yixuan Sheng, Man Lan, and Yuanbin Wu. ECNU at SemEval-2018 Task 11: Using Deep Learning Method to Address Machine Comprehension Task. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1048–1052, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Pavel Shkadzko. Coherent text generation with limited training data. Master's thesis, Saarland University, 2017.

Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open Mind Common Sense: Knowledge Acquisition from the General Public. In On the move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, pages 1223–1237. Springer, Heidelberg, Berlin, 2002.

Robyn Speer and Catherine Havasi. Representing General Relational Knowledge in ConceptNet 5. In Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012), pages 3679–3686, Istanbul, Turkey, May 2012. European Languages Resources Association (ELRA).

Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17, pages 4444–4451, San Francisco, California, USA, 2017. AAAI Press.

Robert Stalnaker. Common ground. Linguistics and philosophy, 25(5-6):701–721, 2002.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In

Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Niket Tandon, Gerard de Melo, and Gerhard Weikum. WebChild 2.0 : Fine-Grained Commonsense Knowledge Distillation. In Proceedings of ACL 2017, System Demonstrations, pages 115–120, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. NewsQA: A Machine Comprehension Dataset. In Proceedings of the 2nd Workshop on Representation Learning for NLP, pages 191–200, Vancouver, Canada, August 2017. Association for Computational Linguistics.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London mathematical society, 2(1):230–265, 1937.

Alan Mathison Turing. Computing Machinery and Intelligence. In Mind, 1950.

Fiona J. Tweedie and R. Harald Baayen. How Variable May a Constant Be? Measures of Lexical Richness in Perspective. Computers and the Humanities, 32(5):323–352, 1998.

F. L. van Holthoon and David R. Olson. Common sense: the foundations for social science, volume 6. University Press of America, 1987.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc., 2017.

Liang Wang, Meng Sun, Wei Zhao, Kewei Shen, and Jingming Liu. Yuanfudao at SemEval-2018 Task 11: Three-way Attention and Relational Knowledge for Commonsense Machine Comprehension. In Proceedings of The 12th International Workshop on Semantic

Evaluation, pages 758–762, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Lilian Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. Inducing Script Structure from Crowdsourced Event Descriptions via Semi-Supervised Clustering. In Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics, pages 1–11, Valencia, Spain, April 2017. Association for Computational Linguistics.

Lilian D. A. Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. DeScript: A Crowdsourced Database of Event Sequence Descriptions for the Acquisition of High-quality Script Knowledge. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), pages 3494–3501, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

Lilian Diana Awuor Wanzare, Michael Roth, and Manfred Pinkal. Detecting Everyday Scenarios in Narrative Texts. In Proceedings of the Second Workshop on Storytelling, pages 90–106, Florence, Italy, August 2019. Association for Computational Linguistics.

Noah Weber, Leena Shekhar, Niranjan Balasubramanian, and Nathanael Chambers. Hierarchical Quantized Representations for Script Generation. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 3783–3792, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. arXiv preprint arXiv:1410.3916, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. arXiv preprint arXiv:1502.05698, 2015.

Jiangnan Xia. Jiangnan at SemEval-2018 Task 11: Deep Neural Network with Attention Method for Machine Comprehension Task. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1063–1067, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Bishan Yang and Tom Mitchell. Leveraging Knowledge Bases in LSTMs for Improving Machine Reading. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1436–1446, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv preprint arXiv:1906.08237, 2019.

Alexander Yeh. More accurate tests for the statistical significance of result differences. In Proceedings of the 17th International Conference on Computational Linguistics, volume 2, pages 947–953, Saarbrücken, Germany, 2000.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. WebAnno: A Flexible, Web-based and Visually Supported System for Distributed Annotations. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 1–6, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. arXiv preprint arXiv:1804.09541, 2018.

Hang Yuan, Jin Wang, and Xuejie Zhang. YNU-HPCC at Semeval-2018 Task 11: Using an Attention-based CNN-LSTM for Machine Comprehension using Commonsense Knowledge. In Proceedings of The 12th International Workshop on Semantic Evaluation, pages 1058–1062, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 93–104, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. ReCoRD: Bridging the Gap between Human and Machine Commonsense Reading Comprehension. arXiv preprint arXiv:1810.12885, 2018.

# Appendices

# A First Version of Event and Participant Annotation for InScript

**Event annotation:**

1. Select the script scenario and look at the attached template.
2. Read the text.
3. For each verb in the text except for modal and auxiliary verbs assign the label according to the rules in Table 1. Note: the examples in the table below show only the result of event annotation, participant annotation will be explained in the next section.

| **label** | **condition** | **example (restaurant scenario)** |
|---|---|---|
| Evoking | Evoking = script evoking element. It activates the script without explicitly referring to any event from the script template. Note: there can be more than one script evoking element in text. | 1. My friend and I have a nice tradition of **going to the restaurant**$_{Evoking}$ once a month. 2. Let me tell you about our last **restaurant visit**$_{Evoking}$. |
| ScrEv_(event_type) | ScrEv = script event, there should be direct correspondence between the verb and the script event from the attached template. | 3. Three friends and I ordered$_{ScrEv_order}$ some sandwiches and soft drinks. 4. The waiter came$_{ScrEv_waiter_comes}$ to our table. 5. I left$_{ScrEv_leave_tip}$ no tip. |

| | | |
|---|---|---|
| MissScrEv | MissScrEv = missing script event. The verb refers to a script event but you don' t find it in the template event list. | 6. The hostess took$_{MissScrEv}$ my coat.<br>7. They were cooking$_{MissScrEv}$ our pizzas for more than one hour. |
| RelNScrEv | RelNScrEv = related non script event, the verb depicts an event which can happen in the script and semantically related to it. Note: the verb "rang" in (ex. 13) is not a script related event since it's not specific to the restaurant scenario and can happen anywhere. In contrast the verb "spilt" in (ex. 9) or "burnt" in (ex. 10) are script related. | 8. I had been dying$_{UnrelEv}$ to try$_{RelNScrEv}$ that restaurant for years.<br>9. The waitress accidentally spilt$_{RelNScrEv}$ the soup on my dress.<br>10. It was so hot that I burnt$_{RelNScrEv}$ my tongue. |
| UnrelEv | UnrelEv = unrelated event, the verb is not related to the script. | 11. I had been dying$_{UnrelEv}$ to try$_{RelNScrEv}$ that restaurant for years.<br>12. I was wondering$_{UnrelEv}$ if I should go$_{ScrEv_get_restaurant}$ to that restaurant.<br>13. I was waiting$_{ScrEv_wait}$ for my order when my phone rang$_{UnrelEv}$. |
| ??? | You don't know. | |

**Table 1:** Event Annotation.

**Participant annotation:**

1. Annotate all arguments of the verbs with ScrEv, MissScrEv or RelNScrEv labels according to the rules in the table below. Important: type of the verb doesn't determine type of the argument. It means that verbs can combine with different types of participants. Compare ex. (10) and (13) in Table 2.

2. Arguments/modifiers of Evoking elements should be annotated as ScrPart_(participant type) or MissScrPart (ex. 4).

3. Note all annotated script participants (ScrPart, MissScrPart or RelNScrPart) and read the text one more time, annotating all occurrences of these participants (also if the verb is not a script event). Look at the examples (5) and (6) in Table 2.

| **label** | **condition** | **example (restaurant scenario)** |
|---|---|---|
| ScrPart_ (participant_type) | ScrPart = script participant, there is a direct correspondence with a participant from the list in the attached template. This label can also be assigned to the argument of the non script event (ex. 4-6). | 1. **We**$_{ScrPart\_customer}$ **had**$_{ScrEv\_eat}$ our **seafood meals**$_{ScrPart\_food}$ and **crème brulees**$_{ScrPart\_food}$.<br>2. The **waitress**$_{ScrPart\_waiter}$ **brought**$_{MissScrEv}$ our **drinks**$_{ScrPart\_drink}$.<br>3. **I**$_{ScrPart\_customer}$ **left**$_{ScrEv\_leave\_tip}$ no **tip**$_{ScrPart\_tip}$.<br>4. **Let**$_{UnrelEv}$ **me**$_{ScrPart\_customer}$ **tell**$_{UnrelEv}$ you about **our**$_{ScrPart\_customer}$ last **restaurant visit**$_{Evoking}$.<br>5. **I**$_{ScrPart\_Customer}$ **live**$_{UnrelEv}$ in New York. ... **I**$_{ScrPart\_customer}$ **ordered**$_{ScrEv\_order}$ **steak**$_{ScrPart\_food}$. |

| | | |
|---|---|---|
| | | 6. A small Italian **restaurant**$_{ScrPart\_restaurant}$ **is**$_{UnrelEv}$ not so far from my house. **I**$_{ScrPart\_customer}$ **went**$_{ScrEv\_get\_restaurant}$ **there**$_{ScrPart\_restaurant}$ **yesterday**$_{MissScrPart}$. |
| MissScrPart | MissScrPart = missing script participant. It's relevant for the script but is missing in the participant list. This label can also be assigned to the argument of the non script event (ex. 9-10). | 7. The **owner**$_{MissScrPart}$ **thanked**$_{RelNScrEv}$ **us**$_{ScrPart\_customer}$ for **choosing**$_{RelNScrEv}$ his **restaurant**$_{ScrPart\_restaurant}$.<br>8. **I**$_{ScrPart\_customer}$ **got**$_{MissScrEv}$ a **coupon**$_{MissScrPart}$ for a free **meal**$_{ScrPart\_food}$ at that **restaurant**$_{ScrPart\_restaurant}$.<br>9. **I**$_{ScrPart\_customer}$ could **see**$_{UnrelEv}$ the **cook**$_{MissScrPart}$ through the window.<br>10. The **dish**$_{MissScrPart}$ **was**$_{UnrelEv}$ dirty. |
| SuppVComp | SuppVComp = support verb complement is a nominal element in such constructions as: take time, be in a rush, give explanation. | 11. The **restaurant**$_{ScrPart\_restaurant}$ **had**$_{RelNScrEv}$ some problems$_{SuppVComp}$ with **air-conditioning**$_{NPart}$.<br>12. The **manager**$_{MissScrPart}$ **gave**$_{SuppVComp}$ **us**$_{ScrPart\_customer}$ no **explanations**$_{SuppVComp}$ for the incident. |

| | | |
|---|---|---|
| NPart | NPart = unrelated referring expression, no relation to the script. | 13. The **waiter**$_{ScrPart\_waiter}$ **came**$_{ScrEv\_waiter\_comes}$ to **us**$_{ScrPart\_customer}$ with an angry **expression**$_{NPart}$ on his face. 14. **I**$_{ScrPart\_customer}$ **gave**$_{ScrEv\_give\_tip}$ the **server**$_{ScrPart\_waiter}$ some **cash**$_{ScrPart\_tip}$ and **my**$_{ScrPart\_customer}$ **visit card**$_{NPart}$. |
| ??? | You don't know. | |

**Table 2:** Participant Annotation.

**Annotation span:**

For events:

Annotate phrasal verbs as a whole (ex. 1-2).

1. Then **I**$_{ScrPart\_customer}$ **waited for**$_{ScrEv\_wait}$ the **waiter**$_{ScrPart\_waiter}$ to **come**$_{ScrEv\_waiter\_comes}$ with our **bill**$_{ScrPart\_bill}$.
2. **We**$_{ScrPart\_customer}$ **looked**$_{ScrEv\_look\_menu}$ through the **menu**$_{ScrPart\_menu}$.

For participants:

Annotate only heads in noun phrases. If they are complex (embedded) noun phrases, annotate the constituents with participant roles separately (ex. 3-4).

3. One of the **waitresses**$_{ScrPart\_waiter}$ **working**$_{RelNScrEv}$ **there**$_{ScrPart\_restaurant}$ **called**$_{RelNScrEv}$ the **manager**$_{MissScrPart}$.
4. My **father**$_{ScrPart\_customer}$ was **holding**$_{RelNScrEv}$ a **glass**$_{MissScrPart}$ of **water**$_{ScrPart\_drink}$.

The compounds should be annotated as a whole.

5. Being a vegetarian **I**$_{ScrPart\_customer}$ couldn't **order**$_{ScrEv\_order}$ **lasagne bolognese**$_{ScrPart\_food}$.

6. **I**$_{ScrPart\_customer}$ **intended**$_{UnrelEv}$ to **pay**$_{ScrEv\_pay\_bill}$ with the **credit card**$_{ScrPart\_payment\_method}$.

**General Notes:**

Try to be consistent in your annotation. Annotate the synonyms and words with the same semantic role identically (ex. 7-10).

7. (event annotation: went/drove)
   **I**$_{ScrPart\_customer}$ **went**$_{ScrEv\_get\_restaurant}$ to the **restaurant**$_{ScrPart\_restaurant}$ on foot.
   **My**$_{ScrPart\_customer}$ **daughter**$_{NPart}$ **drove**$_{ScrEv\_get\_restaurant}$ **me**$_{ScrPart\_customer}$ to the **restaurant**$_{ScrPart\_restaurant}$.

8. (event annotation: give/receive)
   **I**$_{ScrPart\_customer}$ **gave**$_{ScrEv\_leave\_tip}$ **her**$_{ScrPart\_waiter}$ a good **tip**$_{ScrPart\_tip}$.
   The **waitress**$_{ScrPart\_waiter}$ **received**$_{ScrEv\_leave\_tip}$ a good **tip**$_{ScrPart\_tip}$ from **me**$_{ScrPart\_customer}$.

9. (participant annotation: waiter/man)
   The **waiter**$_{ScrPart\_waiter}$ **appeared**$_{ScrEv\_waiter\_comes}$.
   The bald **man**$_{ScrPart\_waiter}$ completely **dressed**$_{UnrelEv}$ in black was apparently **annoyed**$_{UnrelEv}$.

10. (participant annotation: Karen/wife, cheesecake/dessert)
    **My**$_{ScrPart\_customer}$ **wife**$_{ScrPart\_customer}$ **ordered**$_{ScrEv\_order}$ a **cheesecake**$_{ScrPart\_food}$.
    **Karen's**$_{ScrPart\_customer}$ **dessert**$_{ScrPart\_food}$ **was**$_{UnrelEv}$ delicious.

**Coreference annotation:**

**General notes:**

Annotate all participants (nouns + pronouns) in text which refer to the same object. Note: annotate only if there is a chain (more than one mention in text). You should assign the label only to the head in NP (ex. 12). There are also some cases like (ex. 13) when the whole NP (my wife and I) represents a coreference chain and its constituents build their own coreference chains (my wife and I correspondingly). In such cases you should annotate them separately. If there are script participants in the coreference chain make sure that all coreferring elements have the same role (ex. 17-18).

Coreference chain:

waiter = the bald man completely dressed in black (coref_1)

11. The [**waiter**$_{ScrPart\_waiter}$] **appeared**$_{ScrEv\_waiter\_comes}$.
12. The bald [**man**$_{ScrPart\_waiter}$] completely **dressed**$_{UnrelEv}$ in black was apparently **annoyed**$_{UnrelEv}$.

Coreference chains:

wife = she (coref_1)

my = I (coref_2)

wife and I = we (coref_3)

13. [**My**$_{ScrPart\_customer}$]$_{coref\_2}$ [[**wife**$_{ScrPart\_customer}$]$_{coref\_1}$ and [**I**$_{ScrPart\_customer}$]$_{coref\_2}$]$_{coref\_3}$ **went to the restaurant**$_{Evoking}$.
14. [**She**$_{ScrPart\_customer}$]$_{coref\_1}$ **ordered**$_{ScrEv\_order}$ the **dessert**$_{ScrPart\_customer}$.
15. [**I**$_{ScrPart\_customer}$]$_{coref\_2}$ was **surprised**$_{UnrelEv}$ with the service.
16. [**We**$_{ScrPart\_customer}$]$_{coref\_3}$ **decided**$_{UnrelEv}$ not to **leave**$_{ScrEv\_leave\_tip}$ a **tip**$_{ScrPart\_tip}$.

Coreference chains:

waiter = his (coref_1)

me = I = a waitress = myself (coref_2)

17. That [**waiter**$_{ScrPart\_waiter}$]$_{coref\_1}$ **was**$_{UnrelEv}$ very rude to [**me**$_{ScrPart\_customer}$]$_{coref\_2}$.

18. [**I**$_{ScrPart\_customer}$]$_{coref\_2}$ **have**$_{UnrelEv}$ an experience as **working**$_{UnrelEv}$ as a [**waitress**$_{ScrPart\_customer}$]$_{coref\_2}$ [**myself**$_{ScrPart\_customer}$]$_{coref\_2}$ and was really **irritated**$_{UnrelEv}$ by [**his**$_{ScrPart\_waiter}$]$_{coref\_1}$ behaviour.

**Example of the scenario template: restaurant visit**

**Events:**

| **Script Event** | **WebAnno Label** |
|---|---|
| get to restaurant | get_restaurant |
| take a seat | take_seat |
| look at menu | look_menu |
| waiter comes | waiter_comes |
| order | order |
| wait for food | wait |
| spend time at the restaurant | spend_time_rest |
| eat | eat |
| pay bill | pay |
| leave tip | leave_tip |
| leave restaurant | leave |

**Participants:**

| | | |
|---|---|---|
| restaurant | customer | waiter |
| table | menu | drink |
| food | bill | payment method |
| receipt | tip | |

# B Final Version of Event and Participant Annotation Guidelines for InScript

## Event annotation:

This section explains the event annotation.

1. Select the script scenario and look at the attached template.
2. Read the text.
3. For each verb in the text **except for auxiliary verbs** assign the label according to the rules in Table 1. Note: the examples in the table below show only the result of the event annotation, participant annotation will be explained in the next section.

   **Note:** In the following examples, all annotations for events and participants are given. The actually relevant labeled parts are highlighted in **bold red**.

| **label** | **condition** | **example (restaurant scenario)** |
|---|---|---|
| Evoking | Evoking = script evoking element. It activates the script without explicitly referring to any event from the script template. Evoking elements may also be expressed by event-denoting nouns (ex. 2). Note: there can be more than one script evoking element in the text. | 1. My friend and I have a nice tradition of **going to the restaurant**$_{\text{Evoking}}$ once a month.<br>2. Let me tell you about our last restaurant **visit**$_{\text{Evoking}}$. |

| | | |
|---|---|---|
| ScrEv_ (event_type) | ScrEv = script event. There should be a direct correspondence between the verb and a script event from the attached template. | 3. Three friends and I **ordered**$_{\text{ScrEv\_order}}$ some sandwiches and soft drinks.<br>4. The waiter **came**$_{\text{ScrEv\_waiter\_comes}}$ to our table.<br>5. I **left**$_{\text{ScrEv\_leave\_tip}}$ no tip. |
| ScrEv_other | ScrEv_other = other script event. The verb refers to a script event which is not included in the template event list. This means that the event is too fine-grained or infrequent to have its own label. However, it still clearly belongs to the script. | 6. The hostess **took**$_{\text{ScrEv\_other}}$ my coat.<br>7. They were **cooking**$_{\text{ScrEv\_other}}$ our pizzas for more than one hour. |
| RelNScrEv | RelNScrEv = related non script event. The verb depicts an event which can happen in the script and is semantically related to it. Note: the verb "rang" in (ex. 13) is not a script related event since it's not specific to the restaurant scenario | 8. I had been dying$_{\text{UnrelEv}}$ to **try**$_{\text{RelNScrEv}}$ that restaurant for years.<br>9. The waitress accidentally **spilled**$_{\text{RelNScrEv}}$ the soup on my dress.<br>10. It was so hot that I **burnt**$_{\text{RelNScrEv}}$ my tongue. |

| | | |
|---|---|---|
| | and can happen anywhere. In contrast the verb "spilled" in (ex. 9) or "burnt" in (ex. 10) are script related. | |
| UnrelEv | UnrelEv = unrelated event. The verb is not related to the script. | 11. I had been **dying**$_{\text{UnrelEv}}$ to try$_{\text{RelNScrEv}}$ that restaurant for years.<br>12. I was **wondering**$_{\text{UnrelEv}}$ if I should go$_{\text{ScrEv_get_restaurant}}$ to that restaurant.<br>13. I was waiting$_{\text{ScrEv_wait}}$ for my order when my phone **rang**$_{\text{UnrelEv.}}$ |
| Unclear | You don't know. | |

**Table 1. Event annotation.**

## Participant annotation:

This section explains the participant annotation.

1. Annotate all heads of noun phrases, personal and possessive pronouns according to the rules in the table below. Note that verbs can combine with different types of participants.

2. Arguments/modifiers of Evoking elements should be annotated as ScrPart_(participant type) or ScrPart_other (ex. 4).

| label | condition | example (restaurant scenario) |
|---|---|---|
| ScrPart_ (participant_type) | ScrPart = script participant. There is a direct correspondence with a participant from the list in the attached template. | 1. We$_{\text{ScrPart_customer}}$ had$_{\text{ScrEv_eat}}$ our$_{\text{ScrPart_customer}}$ seafood meals$_{\text{ScrPart_food}}$ and crème brulées$_{\text{ScrPart_food}}$.<br>2. The waitress$_{\text{ScrPart_waiter}}$ brought$_{\text{ScrEv_other}}$ our$_{\text{ScrPart_customer}}$ drinks$_{\text{ScrPart_drink}}$.<br>3. I$_{\text{ScrPart_customer}}$ left$_{\text{ScrEv_leave_tip}}$ no tip$_{\text{ScrPart_tip}}$.<br>4. Let$_{\text{UnrelEv}}$ me$_{\text{ScrPart_customer}}$ tell$_{\text{UnrelEv}}$ you$_{\text{NPart}}$ about our$_{\text{ScrPart_customer}}$ last restaurant visit$_{\text{Evoking, ScrPart_other}}$. |

| | | |
|---|---|---|
| | | 5. I$_{\text{ScrPart_customer}}$ live$_{\text{UnrelEv}}$ in *New York*$_{\text{NPart}}$. ... I$_{\text{ScrPart_customer}}$ ordered$_{\text{ScrEv_order}}$ a steak$_{\text{ScrPart_food}}$.<br>6. A small Italian restaurant$_{\text{ScrPart_restaurant}}$ is$_{\text{UnrelEv}}$ not so far from my$_{\text{ScrPart_customer}}$ house$_{\text{NPart}}$. I$_{\text{ScrPart_ustomer}}$ went$_{\text{ScrEv_get_restaurant}}$ there$_{\text{ScrPart_restaurant}}$ yesterday$_{\text{ScrPart_other}}$. |
| ScrPart_other | ScrPart_other = other script participant. This participant is relevant for the script but is not in the participant list. This means that the participant is too fine-grained or infrequent to have its own label. However, it still clearly belongs to the script. | 7. The owner$_{\text{ScrPart_other}}$ thanked$_{\text{RelNScrEv}}$ us$_{\text{ScrPart_customer}}$ for choosing$_{\text{RelNScrEv}}$ his restaurant$_{\text{ScrPart_restaurant}}$.<br>8. I$_{\text{ScrPart_customer}}$ got$_{\text{ScrEv_other}}$ a coupon$_{\text{ScrPart_other}}$ for a free meal$_{\text{ScrPart_food}}$ at that restaurant$_{\text{ScrPart_restaurant}}$.<br>9. I$_{\text{ScrPart_customer}}$ could see$_{\text{UnrelEv}}$ the cook$_{\text{ScrPart_other}}$ through the window$_{\text{NPart}}$.<br>10. The dish$_{\text{ScrPart_other}}$ was$_{\text{UnrelEv}}$ dirty. |

| | | |
|---|---|---|
| SuppVComp | SuppVComp = support verb complement. This is a nominal argument in support verb constructions as: take time, be in a rush, give explanation. **Important Note:** The support verb should in this case get a meaningful event label that describes the event spanned by the complete construction. **This is the only case in which auxiliaries and modals are labeled.** | 11. The restaurant$_{\text{ScrPart_restaurant}}$ had$_{\text{RelNScrEv}}$ some **problems**$_{\text{SuppVComp}}$ with air-conditioning$_{\text{NPart}}$.<br>12. The manager$_{\text{ScrPart_other}}$ gave$_{\text{RelNScrEv}}$ us$_{\text{ScrPart_customer}}$ no **explanation**$_{\text{SuppVComp}}$ for the incident$_{\text{NPart}}$. |
| Head_of_Partitive | Head_of_Partitive = head of a partitive-like construction. This label marks the noun head in a partitive phrase. | 13. Jane$_{\text{ScrPart_customer}}$ ordered$_{\text{ScrEv_order}}$ a **glass**$_{\text{Head\_of\_Partitive}}$ of wine$_{\text{ScrPart_drink}}$.<br>14. I$_{\text{ScrPart_customer}}$ really liked$_{\text{RelNScrEv}}$ that **kind**$_{\text{Head\_of\_Partititve}}$ of bread$_{\text{ScrPart_food}}$. |
| No_label | No_label. No_label is used for non - referential time expressions, expletive it (ex. 16), idioms (ex. 16) and some other abstracta (ex. 15 and 17). | 15. Every **time**$_{\text{No\_label}}$ I$_{\text{ScrPart_customer}}$ eat$_{\text{ScrEv_eat}}$ there$_{\text{ScrPart_restaurant}}$ I$_{\text{ScrPart_customer}}$ leave$_{\text{ScrEv_leave_tip}}$ a big tip$_{\text{ScrPart_tip}}$ for the waiter$_{\text{ScrPart_waiter}}$. |

| | | |
|---|---|---|
| | | 16. It$_{\text{No_label}}$ was my$_{\text{ScrPart_customer}}$ turn$_{\text{No_label}}$ to pay$_{\text{ScrEv_pay}}$ the bill$_{\text{ScrPart_bill}}$.<br>17. This restaurant$_{\text{ScrPart_restaurant}}$ was a lot$_{\text{No_label}}$ better since it$_{\text{ScrPart_restaurant}}$ wasn't that expensive and had very friendly staff$_{\text{ScrPart_other}}$ . |
| NPart | NPart = non-participant. This is a referring expression without any relation to the script. | 18. The waiter$_{\text{ScrPart_waiter}}$ came$_{\text{ScrEv_waiter_comes}}$ to us$_{\text{ScrPart_customer}}$ with an angry expression$_{\text{No_label}}$ on his$_{\text{ScrPart_waiter}}$ face$_{\text{NPart}}$.<br>19. I$_{\text{ScrPart_customer}}$ gave$_{\text{ScrEv_give_tip}}$ the server$_{\text{ScrPart_waiter}}$ some cash$_{\text{ScrPart_tip}}$ and my$_{\text{ScrPart_customer}}$ business card$_{\text{NPart}}$. |
| Unclear | You don't know. | |

**Table 2. Participant annotation.**

## Annotation span

In this section, we explain some special cases for annotation spans.

**For events:**

Annotate phrasal verbs as a whole (ex. 1-2). If this is not possible because the verb and particle are not next to each other, just annotate the main verb.

1. Then I$_{\text{ScrPart_customer}}$ **waited for**$_{\text{ScrEv\_wait}}$ the waiter$_{\text{ScrPart_waiter}}$ to come$_{\text{ScrEv_waiter_comes}}$ with our$_{\text{ScrPart_customer}}$ bill$_{\text{ScrPart_bill}}$.
2. We$_{\text{ScrPart_customer}}$ **looked through**$_{\text{ScrEv\_look\_menu}}$ the menu$_{\text{ScrPart_menu}}$.

For Evoking elements, annotate the complete phrase that evokes the script. Note that there can be other events and participants within the evoking element. (cf.ex. 3)

3. We$_{\text{ScrPart_customer}}$ ***went***$_{\text{ScrEv\_get\_restaurant}}$ **to the *restaurant***$_{\text{ScrPart\_restaurant}}$$_{\text{Evoking}}$ that day.

**For participants:**

- For complex (embedded) noun phrases, annotate the constituents with participant roles separately (ex. 4-5).

  4. One of the waitresses$_{\text{ScrPart_waiter}}$ working$_{\text{RelNScrEv}}$ there$_{\text{ScrPart_restaurant}}$ called$_{\text{RelNScrEv}}$ the manager$_{\text{ScrPart_other}}$.

  5. My father$_{\text{ScrPart_customer}}$ was holding$_{\text{RelNScrEv}}$ a glas$_{\text{ScrPart_other}}$ of water$_{\text{ScrPart_drink}}$.

- Compounds should be annotated as a whole (ex. 6-7). In general, you don't need to do nested annotations (e.g. in 7, you don't need to annotate *card* separately). **However, for some foreign noun compounds, in which the last word is not the semantic head, please annotate twice** (ex. 7)!

  6. I$_{\text{ScrPart_customer}}$ intended$_{\text{UnrelEv}}$ to pay$_{\text{ScrEv_pay_bill}}$ with the **credit card**$_{\text{ScrPart\_payment\_method}}$.

  7. Being a vegetarian I$_{\text{ScrPart_customer}}$ couldn't order$_{\text{ScrEv_order}}$ **[lasagne**$_{\text{ScrPart\_food}}$ **bolognese]**$_{\text{ScrPart\_food}}$.

- Enumerations are annotated twice. You should annotate both the elements of the enumeration and the whole enumeration with a participant type label, if possible.

  8. I ordered [a *steak*$_{\text{ScrPart\_food}}$, *fries*$_{\text{ScrPart\_food}}$, and a *coke*$_{\text{ScrPart\_food}}$]$_{\text{ScrPart\_food}}$.

**General notes:**

Try to be consistent in your annotation. Annotate synonyms and words with the same semantic role identically (ex. 9-12).

9. (event annotation: went/drove)

I$_{\text{ScrPart_customer}}$ went$_{\text{ScrEv_get_restaurant}}$ to the restaurant$_{\text{ScrPart_restaurant}}$ on foot.

My$_{\text{ScrPart_customer}}$ daughter$_{\text{NPart}}$ drove$_{\text{ScrEv_get_restaurant}}$ me$_{\text{ScrPart_customer}}$

to the restaurant$_{\text{ScrPart_restaurant}}$.

10. (event annotation: give/receive)

I$_{\text{ScrPart_customer}}$ gave$_{\text{ScrEv_leave_tip}}$ her$_{\text{ScrPart_waiter}}$ a good tip$_{\text{ScrPart_tip}}$.

The waitress$_{\text{ScrPart_waiter}}$ received$_{\text{ScrEv_leave_tip}}$ a good tip$_{\text{ScrPart_tip}}$ from me$_{\text{ScrPart_customer}}$.

11. (participant annotation: waiter/man)

The waiter$_{\text{ScrPart_waiter}}$ appeared$_{\text{ScrEv_waiter_comes}}$.

The bald man$_{\text{ScrPart_waiter}}$ completely dressed$_{\text{UnrelEv}}$ in black was apparently annoyed$_{\text{UnrelEv}}$.

12. (participant annotation: Karen/wife, cheesecake/dessert)

My$_{\text{ScrPart_customer}}$ wife$_{\text{ScrPart_customer}}$ ordered$_{\text{ScrEv_order}}$ a cheesecake$_{\text{ScrPart_food}}$.

Karen's$_{\text{ScrPart_customer}}$ dessert$_{\text{ScrPart_food}}$ was$_{\text{UnrelEv}}$ delicious.

## Coreference annotation

Annotate all participants (nouns + pronouns) in the text which refer to the same object. Annotate only if there is a chain (more than one mention in the text).

waiter = the bald man completely dressed in black (coref_1)

13. The [waiter$_{\text{ScrPart_waiter}}$]$_{\text{coref\_1}}$ appeared$_{\text{ScrEv_waiter_comes}}$.

14. The bald [man$_{\text{ScrPart_waiter}}$]$_{\text{coref\_1}}$ completely dressed$_{\text{UnrelEv}}$ in black was apparently annoyed$_{\text{UnrelEv}}$.

In some cases, there are group referents that contain several single entities. In the annotation tool, you will find 2 coreference labels for this case: *single* and *group*. An example is given in 15-17. *We* must be connected to both *wife* and *I*. These two links should then be labeled with *group*, i.e. coref_3 is a *group* chain. **Note that the direction of the link is of importance! You have to draw a link from the members to the group entity, not vice versa.**
If you cannot find a single participant label for the group entity, use the *Unclear* label. [1]

wife = she (coref_1); my = I (coref_2); wife = I = we (coref_3)

15. [My$_{\text{ScrPart_customer}}$]$_{\text{coref\_2}}$ [wife$_{\text{ScrPart_customer}}$]$_{\text{coref\_1}}$ ordered$_{\text{ScrEv_order}}$the dessert$_{\text{ScrPart_food}}$ after
[she$_{\text{ScrPart_customer}}$]$_{\text{coref\_1}}$ looked at the menu.

16. [I$_{\text{ScrPart_customer}}$]$_{\text{coref\_2}}$ was surprised$_{\text{UnrelEv}}$ with the really bad service.

17. [We$_{\text{ScrPart_customer}}$]$_{\text{coref\_3}}$ decided$_{\text{UnrelEv}}$ not to leave$_{\text{ScrEv_leave_tip}}$ a tip$_{\text{ScrPart_tip}}$.

There are also complex group entities as in ex. 18 in which the whole NP ("my wife and I") represents a coreference chain and its constituents build their own chains, too. In such cases you should annotate them separately.

wife = she (coref_1); my = I (coref_2); wife = I = we (coref_3)

18. [My$_{\text{ScrPart_customer}}$]$_{\text{coref\_2}}$ [wife$_{\text{ScrPart_customer}}$]$_{\text{coref\_1}}$ and [I$_{\text{ScrPart_customer}}$]$_{\text{coref\_2}}$ went to the restaurant$_{\text{Evoking}}$.

[1] However, there are scenarios in which we added a heterogeneous participant label. An example is the *haircut* scenario. There, the group entity "I and the hairdresser" is mentioned very often, so we introduced a participant label for it.

19. [She$_{\text{ScrPart_customer}}$]$_{\text{coref\_1}}$ ordered$_{\text{ScrEv_order}}$ the dessert$_{\text{ScrPart_food}}$.
20. [I$_{\text{ScrPart_customer}}$]$_{\text{coref\_2}}$ was surprised$_{\text{UnrelEv}}$ with the service.
21. [We$_{\text{ScrPart_customer}}$]$_{\text{coref\_3}}$ decided$_{\text{UnrelEv}}$ not to leave$_{\text{ScrEv_leave_tip}}$ a tip$_{\text{ScrPart_tip}}$.

## Example of the scenario template: restaurant visit

**Events:**

| **Script event** | **WebAnno label** |
|---|---|
| get to restaurant | get_restaurant |
| take a seat | take_seat |
| look at menu | look_menu |
| waiter comes | waiter_comes |
| order | order |
| wait for food | wait |
| spend time at the restaurant | spend_time_rest |
| eat | eat |
| pay bill | pay |
| leave tip | leave_tip |
| leave restaurant | leave |

**Participants:**

restaurant

customer

waiter

table

menu

drink

food

bill

payment method

receipt

tip

# C Lexical Entailment Annotation Guidelines - Events

## C.1 Task

In this study, we want to investigate how abstract script structures are realized and instantiated in narrative texts. To do so, we are going to look at pairs of sentences from narrative texts (Sentence) and abstract descriptions of script events, so called patterns (P).

A pattern is a summary of several single short descriptions of the event (event descriptions, ED). A pattern usually consists of a verb and zero or more placeholders for participants. We will call the head verb of the pattern and any verb in the sentence that describes an event *event verb*.

Participants describe persons, objects or concepts that are involved in a script, the participant placeholders start with *p_*. Optional participants are marked with round brackets "()". An example pattern is **set p_oven (to p_temperature) (according to p_baking-instructions)**. It subsumes for example the following EDs:

- set oven to 360 degrees
- set oven to correct temperature according to box directions
- set oven

In this study, you will compare event verbs and participants between patterns and sentences. To do so, you first have to select the matching pattern, i.e. the one that is most similar to the sentence. Then you have to align event verb and participants between sentence and pattern and assign a label to that alignment.

## C.2 Annotation Material

During the annotation, you are shown one sentence at a time together with the patterns for the corresponding event (as shown in Figure 1).

Colored boxes represent participants and event verbs. The colors can vary from item to item and the same color can be used for several types, so in the beginning make sure you know which colors represent which types.

The set of patterns is sorted and grouped to represent its inner structure:

- Patterns that describe the event in different ways, e.g. from different points of view, are separated in blocks (e.g. lines 04-06, 10-12).
- Patterns that have the same meaning and argument structure are written in consecutive lines (e.g. lines 04, 05, 06).
- Within a block, patterns that are not equivalent are separated by a blank line and sorted according to their specificity with the most specific patterns at the top (e.g. lines 10-12).


**Figure 1:** Annotation view for an instance of the event preheat oven from the cake-scenario. The sentence has to be aligned to line 05.

## C.3 Guidelines for the Manual Annotation

There are two subtasks for this annotation: You first have to select a single pattern that you think matches the sentence best. Then, you have to draw alignment links between matching chunks of a pattern and chunks of a sentence. As sentences are usually much longer than

pattern, in most cases there exists no simple one-to-one alignment. So, you only have to concentrate on the *event verb* and the *participants*.

Note that in the sentence, only one event verb is marked at a time. You only have to look at the marked verb and all of its syntactic dependents, nothing else.

**Task I: Find the Best Pattern for a Text Sentence**

In order to select the most suitable pattern (from a list as in figure 1), first select a *block*. The blocks describe rather different actions (e.g. the action of turning on the oven vs. waiting until it gets hot or eating vs. cutting vs. serving a cake), so this selection should be unambiguous. Many events even have only one block.

Within a block, check the patterns top down and stop as soon as you have found a matching pattern/group of patterns. A pattern matches a sentence if it is as least as general as the sentence.

When several patterns are equivalent (which typically implies that they have synonymous verbs), pick the one that has the same verb as the sentence. If none of the patterns contains the verb take the one that best matches the participants (or the first pattern if there is no difference).

As the verb is the head of a sentence, start by comparing the verbs and then look at the participants.

Sentence 1: I *purchased* all of the ingredients including cake mix, eggs, and strawberries.
Sentence 2: I **grabbed** some eggs and butter out the fridge.
Pattern 1: enter p_store

Pattern 2: go to p_store
—
Pattern 3: *purchase* p_ingredients
Pattern 4: buy p_ingredients
—
Pattern 5: get out p_ingredients
Pattern 6: **take out** p_ingredients (from p_cabinet)

Pattern 7: gather p_ingredients (according to p_baking_instructions)
Pattern 8: assemble p_ingredients
Pattern 9: set p_ingredients

Pattern 10: get p_ingredients from p_cabinet

Pattern 11: prepare p_ingredients according to p_baking_instructions

In this example, sentence 1 gets aligned to pattern 3, sentence 2 gets aligned to pattern 6. Sentence 1 describes buying ingredients, so it has to be part of the second block. As pattern 3 contains the same verb, it is aligned to this pattern. Sentence 1 describes getting ingredients from the cabinet, so it has to be part of the third block. No pattern contains the same verb, but get out and take out are the first verbs of which grab is a Hyponym. As the two patterns are in consecutive lines, they are equivalent for the verb. As pattern 6 contains the same participants as the sentence, we pick this pattern.

**Task II: Find Pairs and Label them**

After you have decided for a pattern, align matching verb-verb and participant-participant-pairs. In most situations this should be unambiguous. One simple example is:

(32) Sentence: I added **milk** to the cake *batter*
Pattern: add **p_ingredients** to *p_dough*

In this example, align "added" to "add", "milk" to "p_ingredients" and "batter" to "p_dough". There are also cases in which one participant type occurs several times. In this case, make sure to connect the participants based on their role to the verb. One example is:

(33) Sentence: I added **milk** to the *eggs*
Pattern: add **p_ingredients** to *p_ingredients*

In this example, you have to align "add" with "added", "milk" with the first "p_ingredients" and "eggs" with the second "p_ingredients".
**Note that you only need to align participants that are dependents of the event verb.** In the next example, imagine we look at the "add" event verb. You do not have to align "batter" to the pattern.

(34) Sentence: I *added* **milk** and stirred the batter well.
Pattern: *add* **p_ingredients**

After drawing the alignment links, you have to label every link. We will explain the labels in the next section.

## C.4 Annotation Labels

**Label Overview**

In this section you can find an overview over all labels that are used together with short descriptions. Examples and more elaborate explanations can be found in the next two sections.

*Verb Labels*

| Label | Short Description |
|---|---|
| *Synonym* | The two verbs are synonyms. |
| *Hyponym* | The verb in the text is more specific. |
| *Hypernym* | The verb in the pattern is more specific (rare case!). |
| *Incorporation* | One verb has the combined meaning of another verb and a participant. |
| *Diathesis* | The verbs describe the same action from different perspectives. |
| *Phrasal Verb* | One of the verbs is a particle verb with the same meaning. |
| *Context-Relatedness* | The verbs are only related when looking at the context. |
| *Inference* | A higher order inference is necessary to link the verbs. |
| *NoMatch* | No match is possible. |

*Participant Labels*

| Label | Short Description |
|---|---|
| *Instantiation* | A noun in the text instantiates a participant placeholder. |
| *Anaphora* | A pronoun in the text instantiates a participant placeholder. |
| *Shared Participant* | A participant is shared between two verbs. |
| *Label Inconsistency* | Two different participant types fill the same role for the verb. |
| *Additional Participant* | A participant in the sentence that is not mentioned in the pattern. |

**Verb Labels**

**Synonym** The same (= includes identity) or a very similar verb is used.

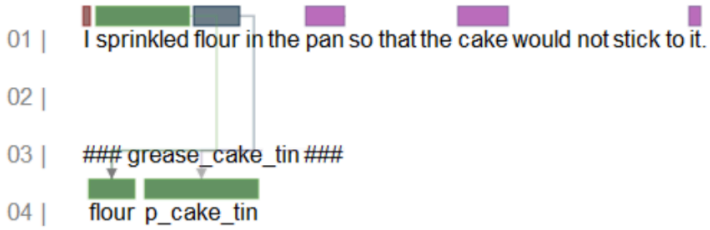
(35) Sentence: Next they slowly **added** the wet ingredients to the dry.
Pattern: **Add** p_ingredients

(36) Sentence: The toothpick came out dry , so I took the cake out and **put** it on a rack to cool.
Pattern: **Place** (p_cake) on p_utensil to cool

**Incorporation [ Two Links ]** A verb has the combined meaning of a verb and one of its participants (typically the direct object). Draw a link from both words to the verb and assign the label Incorporation to both.

(37) Sentence: I **sprinkled flour** in the pan so that the cake would not stick to it.
Pattern: **Flour** p_cake-tin

**!Attention:** For the label *Incorporation* you have to draw two links to the verb in the pattern.




**Hyponym (Text)** The verb in the text is more specific than the verb in the most specific matching pattern.

(38) Sentence: I then **melted** 1/2 cup of butter and 1 ounce of chocolate in a small pan.
Pattern: **Prepare** p_ingredients

**Hypernym (Text)** The verb in the text is less specific than the verb in the pattern (and no more general pattern is matching).

(39) Sentence: I then **prepared** 1/2 cup of butter and 1 ounce of chocolate in a small pan.
Pattern: **melt** p_ingredients

**!Attention:** Don't confuse *Hypernym* and *Hyponym*! We expect *Hypernym* to occur only *very* rarely.

**Diathesis** The two verbs describe the same action from a different view. The most common example is passivization, or verb pairs as *give-receive*, *borrow-lend*.

(40) Sentence: Once the oven **was preheated**, I put the cake in.
Pattern: **Preheat** p_oven

(41) Sentence: I **received** the cake from a friend
Pattern: **Give** p_cake to friend

**Phrasal Verb** The verbs are identical and only differ by a particle that doesn't change the meaning essentially. Note that if the particle changes the meaning of the verb completely (as *look, look up, look after*) you should probably choose another pattern.

(42) Sentence: I **gathered up** the butter, eggs and sugar.
Pattern: **Gather** p_ingredients

**Context-Relatedness** The two verbs are related in the context of the script, i.e. they describe exactly the same action, but there is no clear relation between them[2]. In the example, *use* and *put* are unrelated without context, but they describe the same action when looking at the direct context: "use frosting" and "put frosting on cake" are clearly related.

(43) Sentence: I **put** the frosting on the cake
Pattern: **Use** p_frosting

**Inference** A higher order inference is needed to align the verbs. The verbs describe the same event, but cannot be labeled with any of the other labels. If you use this label on verbs, you do not need to align any participants.

(44) Sentence: I **looked** at the clock in my kitchen
Pattern: **set** p_timer (according to p_baking_instructions)

**NoMatch** Only choose in case absolutely no match is possible (i.e. the wrong event is marked). Link the event verb from the sentence to the first pattern event verb and link nothing else.

(45) Sentence: I wanted to make a chocolate cake for my niece's birthday, so I **looked** through my kitchen cupboards to make sure I had all the ingredients Pattern: **get** p_baking_instructions

[2]More concrete: There is presumably no WordNet relation between "put" and "use".

**Participant Labels**

**Instantiation** A noun is an instance of a participant marker. If a participant is realized several times, align all of them.

(46) Pattern: Add **p_ingredients**
Sentence: I added two **eggs**.
Sentence: I added **milk**, **eggs** and **sugar**.

**Anaphora** A pronoun is an instance of a participant marker. If this occurs with another relation (e.g. shared participant) mark both.

(47) Sentence: After sitting and waiting for several minutes for the bathtub to fill, I turned **it** off and got in
Pattern: Turn off **p_water**

**Shared Participant** The participant is shared between two verbs. In the example *them* is labeled "Anaphora + Shared Participant" because it is the direct object of *frost* and of *decorate*.

(48) Sentence: After the cakes have baked and cooled , I frost and decorate **them**.
Pattern: frost (**p_cake**)

**Label Inconsistency** Another type of participant is realized in the text instead of the type from the pattern, as in the example where *large bowl* is labeled as a *p_utensil*. This mainly occurs with unspecific participant sets (utensils, tools...)

(49) Sentence: I took out a large **bowl**.
Pattern: take out **p_cake_tin**

**Additional Participant [ Span-Label ]** This label applies to participants that are mentioned in the text but missing in the pattern. Note that you only need to look at participants that are syntactic dependents of the verb. Mark only the heads of noun/prepositional phrases. So the label applies to **rack** in the first example, but not to *cake batter* in the second example.

(50) Sentence: Then I let it cool on a **rack** on the counter

Pattern: Let cool (p_cake/p_cake tin) (for p_time)

(51) Sentence: While beating the cake batter, I preheated the oven.

Pattern: Preheat p_oven (to p_temperature) (according to p_baking_instructions)

**!Attention:** The label *Additional Participant* is a Span label, not a link label (e.g. "time" in the example)!


## C.5 Some Additional Remarks

- *Using Head Nouns.* When aligning participants, you generally always take the head of a noun phrase for alignment, not the complete phrase (Example 52 below). Possessive pronouns don't have to be aligned (example 1 below). You can also ignore nouns labeled as *Head_of_Partitive* ("kind" in Example 53) and take their dependents instead. Prepositional phrases that are used as attributes to the actual participant can also be ignored ("colour" in Example 54 below).

  (52) Sentence: Then I let my cake **tin** cool.

  Pattern: Let cool (p_cake/**p_cake_tin**) (for p_time)

  (53) Sentence: I took out an especially large kind of cake **tin**.

  Pattern: take out **p_cake_tin**

  (54) Sentence: I took out a cake **tin** with a nice green colour.

  Pattern: take out **p_cake_tin**

- *Multi-Verb Expressions.* Only look at the head verb of a multi-verb expression in a pattern. In the example the verb of the sentence is *cool*, so you should choose pattern 2

for alignment, not pattern 1. Pattern 1 comes first and also contains *cool*, but that is not the head of the pattern.

(55) Sentence: Then I **cooled** my cake.
Pattern 1: Allow (p_cake) to cool
**Pattern 2**: Cool (p_cake)

- *Label Inconsistency vs. Additional Participant.* There are some cases for which the distinction between these two labels is not trivial. In general, you only need to use *Additional Participant* if the participant in the sentence fills a semantic role that is not filled in the pattern. *Label Inconsistency* should be used if the same role is filled by different participants. In the example below, "degrees" will have to be marked as *Additional Participant* and not as *Label Inconsistency*: "p_time" and "degrees" fill different roles (here *time span* vs *modality*).

(56) Sentence: I preheated the oven to 200 **degrees**.
Pattern: Preheat p_oven for p_time

## C.6 Additional Material

**Example Participants - Baking a Cake**

| p_baking_instructions | p_oven | p_cake_tin | p_time |
|---|---|---|---|
| cake recipe | microwave oven | cake pan | minimum time |
| instructions | oven | baking tin | minutes |
| box directions | stove | cake ring | hour |

| p_cake | p_kitchen/location | p_beneficiary | p_timer |
|---|---|---|---|
| cake | kitchen | family | timer |
| cakes | | friends | |

| p_utensil | p_ingredients | p_decoration | p_dough |
|---|---|---|---|
| mixing bowl | mil | decoration | cake batter |
| electric beater | flour | frosting | cake mixture |
| cooling rack | cake mix | sprinkles | cake dough |
| spoon | cake ingredients | glaze | |
| oven mitts | eggs | icing | |

| p_store | p_cabinet | p_temperature | p_cook |
|---|---|---|---|
| supermarket | pantry | oven temperature | you |
| grocery store | cabinet | degrees | |
| | shelf | | |

# D Lexical Entailment Annotation Guidelines - Participants

*These annotations guidelines were originally given to the annotators in German and translated for the purpose of this thesis. The annotations were done via Google Docs.*

**Goal:** Map participants from the texts to participants in the EDs.

**Examples:** *ScrPart_timer*
in the EDs: "timer"
in the text: "timer", "clock", "it", "alarm"

*ScrPart_root*
in the EDs: "ball", "roots"
in the text: "ball", "bottom", "bulb", "root", "tip", "roots", "room", "base"

**2 Tasks:** Choose the matching participant and assign a respective label.

**Labels:**

- Identity (flour - flour, egg - eggs)
- Synonymy (mixer - beater, extract - essence)
- Hyponymy (rake - tools, magnolia - tree, teaspoon - spoon)
- Hypernymy (food - cake, person - friends)
- Meronym (afternoon - day, white - egg)
- Holonym (house - kitchen)
- Co-Hyponym (confection - cake, knife - fork)
- Inference (result - cake, website - schedule)
- Instance ("number 77" for "bus", "8:30" for time or "Paul" for "friend")

  - This is similar to Hyponymy, but you should use it if not a complete class of objects is mentioned, but rather a single object.

- Pronoun (leave participant field empty - it, which, that, he...)
- Error (leave participant field empty)
  - incomprehensible or incomplete without context ("places" or "area" for "bus stop")
  - typos (recipie or over)
  - wrong labels ("cake batter" labeled as *ingredient*)
  - incomplete participants ("dry", instead of "dry ingredients")

**Approach:** For each listed participant, choose the matching participant from the EDs and a label from the dropdown menu.

**Attention:** There are often several possibilities to choose a participant. The word relations should then be used in the order that is given above. For example, if it is possible to choose between "egg" (Co-Hyponymy) and "ingredient" (Hyponymy) for the noun "salt", you should pick "ingredient". If there are several equivalent options, just pick the first one.
If you are not sure about the meaning of a word, you are allowed to consult a dictionary. If this doesn't help, pick *Error* as a label.