

**LEARNING LOGIC RULES FROM TEXT
USING STATISTICAL METHODS FOR
NATURAL LANGUAGE PROCESSING**

by

MISHAL KAZMI

Submitted to the Graduate School of Engineering and Natural Sciences
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

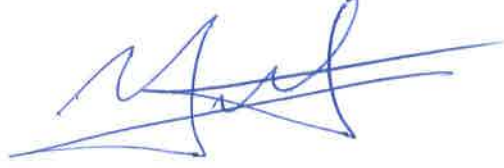
Sabancı University

Spring 2017

LEARNING LOGIC RULES FROM TEXT USING STATISTICAL
METHODS FOR NATURAL LANGUAGE PROCESSING

APPROVED BY:

Prof. Dr. Yücel Saygın
(Thesis Supervisor)



Prof. Dr. Pınar Yolum
(Boğaziçi Üniversitesi)



Asst. Prof. Dr. Hüsnü Yenigün



Asst. Prof. Dr. Murat Can Ganiz
(Marmara Üniversitesi)



Asst. Prof. Dr. Eray Gençay
(Türk Alman Üniversitesi)



DATE OF APPROVAL: 14/06/2017

© Mishal Kazmi 2017
All Rights Reserved

ABSTRACT

Learning Logic rules from text using Statistical methods for Natural Language Processing

Mishal Kazmi

PhD Dissertation, June 2017

Supervisor: Prof. Dr. Yücel Saygın

Co-Supervisor: Asst. Prof. Dr. Peter Schüller

Keywords: *Answer Set Programming, Inductive Logic Programming, Natural Language Processing, Interpretable Semantic Textual Similarity, Sentence Chunking*

The field of Natural Language Processing (NLP) examines how computers can be made to do beneficial tasks by understanding the natural language. The foundations of NLP are diverse and include scientific fields such as electrical and electronic engineering, linguistics, and artificial intelligence. Some popular NLP applications are information extraction, machine translation, text summarization, and question answering.

This dissertation proposes a new methodology using Answer Set programming (ASP) as our main formalism to predict Interpretable Semantic Textual Similarity (iSTS) with a rule-based approach focusing on hard-coded rules for our system, Inspire. We next propose an intelligent rule learning methodology using Inductive Logic Programming (ILP) and modify the ILP-tool eXtended Hybrid Abductive Inductive Learning (XHAIL) in order to test if we are able to learn the ASP-based rules that were hard-coded earlier on the chunking subtask of the Inspire system. Chunking is the identification of short phrases such as noun phrases which mainly rely on Part-of-Speech (POS) tags. We next evaluate our results using real data sets obtained from the SemEval2016 Task-2 iSTS competition to work with a real application which could be evaluated objectively using the test-sets provided by experts.

The Inspire system participated at the SemEval2016 Task-2 iSTS competition in the sub-tasks of predicting chunk similarity alignments for gold chunks and system generated chunks for three different Datasets. The Inspire system extended the basic ideas from SemEval2015

iSTS Task participant NeRoSim, by realising the rules in logic programming and obtaining the result with an Answer Set Solver. To prepare the input for the logic program, the PunktTokenizer, Word2Vec, and WordNet APIs of NLTK, and the Part-of-Speech (POS) and Named-Entity-Recognition (NER) taggers from Stanford CoreNLP were used. For the chunking subtask, a joint POS-tagger and dependency parser were used based on which an Answer Set program determined chunks. The Inspire system ranked third place overall and first place in one of the competition datasets in the gold chunk subtask.

For the above mentioned system, we decided to automate the sentence chunking process by learning the ASP rules using a statistical logical method which combines rule-based and statistical artificial intelligence methods, namely ILP. ILP has been applied to a variety of NLP problems some of which include parsing, information extraction, and question answering. XHAIL, is the ILP-tool we used that aims at generating a hypothesis, which is a logic program, from given background knowledge and examples of structured knowledge based on information provided by the POS-tags.

One of the main challenges was to extend the XHAIL algorithm for ILP which is based on ASP. With respect to processing natural language, ILP can cater for the constant change in how language is used on a daily basis. At the same time, ILP does not require huge amounts of training examples such as other statistical methods and produces interpretable results, that means a set of rules, which can be analysed and tweaked if necessary. As contributions XHAIL was extended with (i) a pruning mechanism within the hypothesis generalisation algorithm which enables learning from larger datasets, (ii) a better usage of modern solver technology using recently developed optimisation methods, and (iii) a time budget that permits the usage of suboptimal results. These improvements were evaluated on the subtask of sentence chunking using the same three datasets obtained from the SemEval2016 Task-2 competition.

Results show that these improvements allow for learning on bigger datasets with results that are of similar quality to state-of-the-art systems on the same task. Moreover, the hypotheses obtained from individual datasets were compared to each other to gain insights on the structure of each dataset. Using ILP to extend our Inspire system not only automates the process of chunking the sentences but also provides us with interpretable models that are useful for providing a deeper understanding of the data being used and how it can be manipulated, which is a feature that is absent in popular Machine Learning methods.

ÖZET

İstatistiksel Yöntemler Kullanarak Doğal Dil İşleme Amacıyla Mantıksal Kural Öğrenmesi

Mishal Kazmi

Doktora Tezi, Haziran 2017

Tez Danışmanı: Prof. Dr. Yücel Saygın

Ortak Tez Danışmanı: Yrd. Doç. Dr. Peter Schüller

Anahtar Kelimeler: *Çözüm Kümesi Programlama, Tümevarım Mantık Programlaması, Doğal Dil İşleme, Yorumlanabilir Anlamsal Metin Benzerliği, Cümle Parçalama*

Doğal Dil İşleme (NLP) alanı bilgisayarların doğal dili anlayarak nasıl yararlı işler yapabileceğini inceler. NLP'nin temelini elektrik ve elektronik mühendisliği, dilbilimi ve yapay zek gibi birçok bilim dalı oluşturur. Bilgi çıkarımı, makine bazlı çeviri, metin özetleme ve soru cevaplama, popüler NLP uygulamalarından bazılarıdır.

Bu tez, Çözüm Kümesi Programlama'yı (ASP) temel alan, Inspire isimli sistemimizde sabit bir şekilde kodlanmış kurallara odaklanan, Yorumlanabilir Anlamsal Metin Benzerlikleri'ni (iSTS) kural bazlı bir yaklaşım ile öngören yeni bir metodoloji önermektedir. Bu metodolojinin yanı sıra, Tümevarımla Mantık Programlama'yı (ILP) kullanarak zeki bir kural öğrenme yöntemi öneriyor ve de Inspire sisteminin parçalama alt görevine sabit bir şekilde girilmiş ASP bazlı kuralları öğrenip öğrenemeyeceğimizi test etmek için bir ILP aracı olan Genişletilmiş Hibrit Dışaçekimsel Tümevarımsal Öğrenme'nin (XHAIL) üzerinde değişiklik yapıyoruz. Parçalama, isim tamlaması gibi kısa sözcük öbeklerinin ağırlıklı olarak Konuşma-Bölümü (POS) etiketleri kullanarak tanımlanmasıdır. Sonuçlarımızı, SemEval2016 Task-2 iSTS yarışmasından elde ettiğimiz gerçek veri setlerini ve uzmanlar tarafından verilmiş test setlerini kullanarak değerlendiriyoruz.

Inspire sistemi, SemEval2016 Task-2 iSTS yarışmasında, üç farklı veri setinde altın parçacıkları ve sistem tarafından üretilen parçalarda yığın benzerliği hizalamalarını tahmin etme alt görevlerinde katıldı. Inspire sistemi, SemEval2015 iSTS katılımcısı NeRoSim'in temel fikir-

lerini, mantık programlamadaki kuralları gerçekleştirerek ve de sonuçları bir Çözüm Kümesi Çözücü ile elde ederek devam ettirdi. Mantık programı için veriyi hazırlarken, PunktTokenizer, Word2Vec ve NLTK'nin WordNet API'ları ve Stanford CoreNLP'nin Konuşma-Bölümü (POS) ve Adlandırılmış Varlık Tanıma (NER) etiketleyicileri kullanıldı. Parçalama alt görevi için, Çözüm Kümesi Programlama tarafından belirlenmiş parçalara dayalı ortak bir POS-etiketleyici ve bağımlılık çözümleyici kullanıldı. Inspire sistemi yarışmada genel kategoride üçüncülüğü, altın parça alt kategorisinde verilerden birinde birinciliği elde etti.

Yukarıda bahsedilen sistem için, ILP adında, kural bazlı ve istatistiksel yapay zek yöntemlerini birleştiren bir istatistiksel mantık metodunu kullanarak ASP kurallarını öğrenmeye ve cümle yığınlama sürecini otomatikleştirmeye karar verdik. ILP yöntemi çözümleyici, bilgi çıkarımı, soru cevaplama gibi çeşitli NLP problemlerine uygulanmıştır. XHAIL, verilen arka plan bilgisinden ve POS etiketleri tarafından sağlanan bilgilere dayanan bir hipotez üretmeyi amaçlayan bir mantık programı olan ILP aracıdır.

Temel zorluklardan biri, XHAIL algoritmasını ASP'yi temel olan ILP için genişletmekti. ILP, doğal dil işlenmesi kapsamında, dilin günlük kullanımdaki sürekli değişimini takip edebilir. Aynı zamanda, ILP diğer istatistiksel yöntemler gibi büyük miktarda eğitim veri setine ihtiyaç duymaz ve yorumlanabilen, analiz edilebilen ve gerektiğinde düzeltilebilen kurallar üretir. Katkı olarak, XHAIL, (i) daha geniş veri setlerinden öğrenmeyi sağlayan, hipotez genelleme algoritması içerisinde bir budama mekanizması, (ii) yakın zamanda geliştirilmiş optimizasyon yöntemlerini kullanarak modern çözücü teknolojisinin daha iyi kullanılması ve (iii) optimal olmayan sonuçların kullanımına izin veren bir zaman bütçesi ile genişletildi. Bu iyileştirmeler, SemEval2016 Task-2 yarışmasından elde edilen aynı üç veri seti kullanılarak, cümle yığınlama alt-görevinde değerlendirildi.

Sonuçlar, bu iyileştirmelerin büyük veri kümelerinde öğrenme konusunda, son teknoloji sistemlerle benzer sonuçlar elde ettiğini göstermiştir. Ayrıca, her bir veri kümesinden elde edilen hipotezler veri kümelerinin yapısını kavrayabilmek için birbirleriyle karşılaştırılmıştır. ILP'yi kullanarak Inspire sistemini genişletmek, yığınlama sürecini otomatikleştirmenin yanı sıra verilerin daha derinleşmesine anlaşılmasını ve nasıl manipüle edilebileceğini yorumlamaya yönelik modeller sağlamıştır; bu da popüler Makine Öğrenme yöntemlerinde bulunmayan bir özelliktir.

This work is dedicated

To My Parents and Sister

My constant support system

To My Husband

My strength and motivation to power through

Acknowledgements

The journey towards completing my dissertation has been quite strenuous and demanding. A majority of people have played a massive role by constantly motivating, supporting, and challenging me in every step of the way. First and foremost I would like to express my deep and sincere gratitude to my supervisors **Prof. Dr. Yücel Saygın** and **Asst. Prof. Dr. Peter Schüller** without them, none of my achievements would have been possible. Their careful editing contributed enormously towards the production of this dissertation.

I am especially indebted to **Asst. Prof. Dr. Peter Schüller** for his patience, time, guidance, and encouragement. His positive outlook and confidence in my research inspired me and gave me confidence to delve deeper into my work.

I would also like to thank the respected jury members **Asst. Prof. Dr. Hüsnü Yenigün**, **Asst. Prof. Dr. Eray Gençay**, **Prof. Dr. Pınar Yolum**, and **Asst. Prof. Dr. Murat Can Ganiz** for their valuable feedback.

I gratefully acknowledge the funding received from Higher Education Commission (HEC) of Pakistan to complete my PhD degree and the grant [114E777] offered by the Scientific and Technological Research Council of Turkey (TUBITAK) to conduct my research.

I am grateful to **Carmine Dodaro** for providing me with support regarding the WASP solver. I would also like to especially thank my friend and fellow colleague **Stefan Rübiger** for providing me with useful coding tips and tricks and for boosting my morale throughout this time.

Lastly, I would like to thank my mother **Sarah Kazmi** for her prayers and unconditional love and support, my father **Faraz Kazmi** for his undying faith in my abilities and his words of wisdom, my husband **Lukas Benz** for inspiring me and standing by my side through thick and thin, and my sister **Sahar Kazmi** for her reassurance and love.

Contents

List of Figures	xi
List of Tables	xii
1 Introduction	1
1.1 Statistical Methods in NLP	2
1.2 Statistical Logical Methods in NLP	3
1.3 Motivation and Scope of this Dissertation	3
1.4 Structure of the Dissertation	5
1.5 Published Work	5
2 Background	7
2.1 Answer Set Programming	7
2.2 Inductive Logic Programming	9
2.2.1 eXtended Hybrid Abductive Inductive Learning	11
3 Materials and Methods	13
3.1 Datasets	13
3.2 Predicting Interpretable Semantic Textual Similarity based on ASP	14

3.2.1	Preprocessing	16
3.2.2	Architecture	18
3.2.2.1	Rule Engine	19
3.2.2.2	NeRoSim Rules	21
3.2.2.3	Interpretation of Answer Sets	26
3.2.2.4	Chunking based on ASP	26
3.3	Chunking based on ILP with ASP	28
3.3.1	Preprocessing	29
3.3.2	Extension of XHAIL	31
3.3.2.1	Kernel Pruning according to Support	32
3.3.2.2	Unsat-core based and Best-effort Optimisation	33
3.3.2.3	Other Improvements	34
4	Experiments	36
4.1	Predicting Interpretable Semantic Textual Similarity based on ASP	36
4.1.1	Run 1	37
4.1.2	Run 2	37
4.1.3	Run 3	38
4.1.4	Scoring	38
4.2	Chunking based on ILP with ASP	39
4.2.1	Model Learning	39
4.2.2	Scoring	41
4.2.3	Evaluation	41

5	Results and Discussion	47
5.1	Predicting Interpretable Semantic Textual Similarity based on ASP	47
5.2	Chunking based on ILP with ASP	48
5.2.1	Training Set and Pruning	49
5.2.2	State-of-the-art comparison	49
5.2.3	Inspection of Hypotheses	50
5.2.4	Impact and Applicability	54
5.2.5	Strengths and weaknesses	55
5.2.6	Design Decisions	55
6	Related Work	57
6.1	Natural Language Processing with Inductive Logic Programming	57
6.2	Systems for Inductive Logic Programming	58
6.2.1	Inductive Learning of Answer Set Programs	58
6.2.2	Incremental Learning of Event Definitions	59
6.2.3	Inspire-ILP	60
7	Conclusion and Future Work	61
	Bibliography	64
	Appendices	71
A	ASP Code	72

List of Figures

2.1	XHAIL architecture	12
3.1	Manual Rules created for Sentence Chunking	27
3.2	General overview of our framework	29
3.3	XHAIL input for the sentence 'Former Nazi death camp guard Demjanjuk dead at 91' from the Headlines Dataset	30
3.4	Modified XHAIL architecture	31

List of Tables

4.1	Dataset partitioning for 11-fold cross-validation experiments	40
4.2	Experimental Results for Headlines Dataset	42
4.3	Experimental Results for Images Dataset	43
4.4	Experimental Results for Answers-Students Dataset	44
5.1	System Performance results	48
5.2	Comparison with systems from SemEval 2016 Task 2	51
5.3	Rules in the best hypotheses obtained from training on 500 sentences for sentence 1 file	52
7.1	Comparison of the benefits and drawbacks provided by Inductive Logic Pro- gramming over standard Machine Learning Techniques and manually-set rules	62

List of Abbreviations and Symbols

NLP	Natural Language Processing
ASP	Answer Set Programming
NAF	Negation as Failure
iSTS	Interpretable Semantic Textual Similarity
ILP	Inductive Logic Programming
XHAIL	eXtended Hybrid Abductive Inductive Learning
ILASP	Inductive Learning of Answer Set Programs
ILED	Incremental Learning of Event Definitions
ML	Machine Learning
NN	Neural Network
SVM	Support Vector Machine
CRF	Conditional Random Field
POS	Part-of-Speech
NER	Named-Entity Recognition

Chapter 1

Introduction

Natural Language Processing (NLP) is a multi-disciplinary research area that focuses on a computer's comprehension of natural language text or speech. NLP researchers gather insight on human understanding to develop systems for computers that can learn these traits of the natural language for multiple tasks such as Machine Translation, Summarization, Information Extraction and Retrieval, and Artificial Intelligence.

In order to understand language, a system must usually first build layers of representation by going to the word-level to gain insight about the morphological structure and nature of the word which usually involves detailed system annotations, then a representation should be made known by going to the sentence-level and determining the overall meaning of the sentence and then finally using world knowledge to determine the context and overall domain. This gives us an overall structured representation of the text. Researchers extract different kinds of information from text for the analysis of NLP systems. In this work, the Stanford Core NLP (Manning et al., 2014) tool is used in order to gather lexical and syntactic information from text.

However, adding more knowledge is not necessarily always useful as it may result in a contradiction, ambiguity, inconsistency or change in a previous belief. This is common in both human reasoning and NLP. We here use Answer Set Programming (ASP) as the formal reasoning language due to its ability to perform common-sense reasoning which includes non-monotonic reasoning (ability to invalidate some conclusions based on the addition of more knowledge) with exceptions. This allows us to formulate such exception based rule formalisms in this work.

1.1 Statistical Methods in NLP

As more knowledge is utilised in structures that represent linguistic expressions, the rule formalisms start getting more rigid. Therefore, using systems purely based on rules proved to be impractical, as it is not possible to give exact and complete characterization to distinguish between a well-formed and an ill-formed utterance. As humans tend to stretch the ‘rules’, NLP researchers found the need to provide some flexibility to account for the neologisms and the informality of natural language. Hence, due to these constant changes in how we use language these days, a need for statistical models arose.

An example of this seen frequently these days is when people say ‘I have to google this information.’ instead of saying they would like to search for this information on the Internet. Therefore a need for statistical models of language emerged. Recently, people have placed great emphasis on this which even led to the field Language Engineering instead of NLP (Manning and Schütze, 1999). They tend to place greater importance to Statistical NLP approaches as they are robust in practice.

One example of a model in NLP for text classification is the bag-of-words model. In this model, a text is represented as a bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model is commonly used in methods of document classification, where the (frequency of) occurrence of each word is used as a feature for training a classifier.

A classifier is a function that maps an input attribute vector to a confidence that the input belongs to a class:

$$f(\vec{x}) = \text{confidence}(\text{class})$$

Some popular classifiers used in NLP include Naive Bayes, Decision Trees, Neural Networks (NNs), Support Vector Machines (SVMs). These are probabilistic so $\text{confidence}(\text{class})$ is a probability distribution (Dumais et al., 1998). Such classifiers associate a vector of fixed length with one of a fixed set of classes. This is a single non-relational knowledge representation without complex internal structure.

1.2 Statistical Logical Methods in NLP

In order to find a middle ground between statistical and logical processing of language, combinations of both paradigms were created. However, purely statistical NLP approaches are better for classification from a small set of alternatives. As mentioned earlier, in NLP tasks we normally require a well-structured representation of the text and are sometimes not provided with large amounts of data for training or annotations. Keeping these notions in mind we here incorporate Inductive Logic Programming (ILP) as our statistical method in order to learn part of the program automatically for our system. Inductive Logic Programming (ILP) uses expressive representation and makes use of logically encoded background knowledge to learn a hypothesis. We include ILP in this work as a means of learning an ASP program that solves an NLP task and investigating the limits of ILP for this task empirically.

1.3 Motivation and Scope of this Dissertation

One of the problems NLP researchers come across when faced with new languages or fastly developing vocabulary is the lack of annotated data available. Acquiring such data takes time and effort. In such cases, hand-coded rules which grasp the linguistic characteristics of the task may be more cost-effective compared to annotating and expecting to obtain similar results indirectly from a Machine Learning (ML) system. This work provides a union of the concepts of ML and rules by using ILP.

We make use of ASP as our main formalism due to its ability to perform common-sense reasoning. By adding more knowledge ASP can add or remove rules for a given ASP program. Our first methodology predicts Interpretable Semantic Textual Similarity (iSTS) using hard-coded rules in ASP for the Inspire system. We next propose learning these hard-coded rules intelligently for the basic subtask in the Inspire system of sentence chunking using ILP. In order to achieve this we present the modifications we made to XHAIL which is our ILP-tool that is being used to learn an ASP rule-based model. We would like to compare whether or not an ILP-based system can outperform or do as good of a job as a ML system without requiring the additional cost of annotations for large amounts of data. We use a small set of training data obtained from the SemEval2016 iSTS Task to develop our proposed

methodology of learning rules to obtain an interpretable solution via ILP in order to chunk sentences. We extend our system, Inspire, with our method in order to evaluate it with real data provided by experts. These results are then compared to other state-of-the-art systems in the sentence chunking subtask which make use of ML or hard-coded rules individually for chunking. We observe that our proposed method of using ILP, which combines logical rules with ML, is able to reach competitive results to the state-of-the-art ML methods in the NLP task of sentence chunking.

The SemEval2016 iSTS Task focuses on systems that are able to explain the similarities and differences between sentence pairs. An additional informative layer is added to formalise chunk alignments present in these sentence pairs. This provides the relation and similarity score of each alignment. The similarity alignment in the Inspire system is based on ideas of SemEval2015 NeRoSim (Banjade et al., 2015) entry, however, we reimplemented the system and realise the rule engine in Answer Set Programming (ASP) (Brewka et al., 2011; Lifschitz, 2008) which gives us flexibility for reordering rules or applying them in parallel. For the chunking subtask, the Inspire system is based on a joint POS-tagger and dependency parser (Bohnet et al., 2013) and an ASP program that determines chunk boundaries. We use ILP to extend the Inspire system by learning this ASP program that determines the chunk boundaries.

Not many applications using ASP-based ILP exist in NLP to the best of our knowledge. Hence, we propose a system incorporating statistical methods such as ILP; capable of learning a program that generates rules which can be utilised by ASP solvers; and to analyse the performance of this technique on a sentence chunking task in NLP. We utilise the datasets provided at SemEval2016 (Agirre et al., 2016). These included three datasets on which we could test our system: Headlines, Images and Answers-Students. For our proposed work we currently had only three tools available to us: Incremental Learning of Event Definitions (ILED) (Katzouris et al., 2015), eXtended Hybrid Abductive Inductive Learning (XHAIL) (Ray, 2009) and Inductive Learning of Answer Set Programs (ILASP) (Law et al., 2015). The ILED and ILASP systems were not feasible for us due to incompatibility (ILED) and scalability (ILASP) issues. XHAIL is the least expressive system but is the most scalable system which also dealt with the negation term in logic programming, which is why we opted to continue our work using the XHAIL system.

We extended XHAIL with the WASP solver which realises modern ASP optimisation al-

gorithms and may result in the utilisation of suboptimal answer-sets by interrupting the solver. Pruning mechanism was also added to XHAIL to learn from larger amounts of data and was evaluated on how it affects results in the chunking task based on how close the results were to the optimal solution.

Our empirical results confirm that our contributions are worthwhile. The addition of pruning allows handling of larger datasets which does not diminish predictive power in most cases. The incorporation of a time budget provides suboptimal solutions that are able to show reasonable predictive power within shorter time frames than otherwise possible.

1.4 Structure of the Dissertation

This Dissertation is organised into six chapters.

Chapter 2: Provides a comprehensive background of the programming language, and tools used.

Chapter 3: Describes in detail the datasets used and the methodologies adapted for each contribution made towards this Dissertation.

Chapter 4: Details the experiments carried out for the Inspire system and for the Chunking task using ILP.

Chapter 5: Mentions the results obtained from the Inspire System and from Chunking with ILP. The results obtained from chunking with ILP are compared to the state-of-the-art systems for this task. We also discuss the results obtained in this chapter.

Chapter 6: Discusses the related work.

Chapter 7: Concludes the findings and discusses possible ideas for extending our work in future research.

1.5 Published Work

The following publications are completely or partially an outcome of this Dissertation:

- Kazmi, Mishal, and Schüller, Peter. "Inspire at SemEval-2016 Task 2: Interpretable semantic textual similarity alignment based on answer set programming." *Proceedings of SemEval (2016)*: 1109-1115.

This paper describes our system, Inspire, for the task on Interpretable Semantic Textual Similarity (Section 3.2).

- Kazmi, Mishal, Schüller, Peter, and Saygin, Yücel. "Improving Scalability of Inductive Logic Programming via Pruning and Best-Effort Optimisation" *Expert Systems with Applications*, 2017 (under review).

This work explains in detail how we extended the chunking subtask of iSTS and automated it for the sentence chunking to be learned automatically (Section 3.3).

- Kazmi, Mishal, and Schüller, Peter. "Best-Effort Inductive Logic Programming via Fine-grained Cost-based Hypothesis Generation." *Machine Learning Journal*, 2017 (under review).

This paper mentions some of the related work done in reference to this Dissertation (Section 6.2.3).

Chapter 2

Background

2.1 Answer Set Programming

A logic program theory normally comprises of an alphabet (variable, constant, etc), vocabulary, and logical symbols (Lloyd, 2012). An ideal of logic programming is for it to be purely declarative so it may express the logic of a computation without a rigid control over the flow description. The popular Prolog (Clocksin and Mellish, 2003) system evaluates rules using resolution, which makes the result of a Prolog program depending on the order of its rules and on the order of the bodies of its rules. Answer Set Programming (ASP) (Brewka et al., 2011; Gebser et al., 2012a) is a more recent logic programming formalism, featuring more declarativity than Prolog by defining semantics based on Herbrand models which are more expressive (Gelfond and Lifschitz, 1988). Hence the order of rules and the order of the body of the rules does not matter in ASP. Most ASP programs follow the Generate-Define-Test structure (Lifschitz, 2002) to (i) generate a space of potential solutions, (ii) define auxiliary concepts, and (iii) test to invalidate solutions using constraints or incurring a cost on non-preferred solutions.

An ASP program consists of rules of the following structure:

$$a \leftarrow b_1, \dots, b_m, \mathbf{not} b_{m+1}, \dots, \mathbf{not} b_n$$

where, a and b_i are atoms from a first-order language, a is the head and $b_1, \dots, \mathbf{not} b_n$ is the body of the rule, and \mathbf{not} is negation as failure. Variables start with capital letters, facts

(rules without body condition) are written as ‘ a .’ instead of ‘ $a \leftarrow$ ’. Intuitively a is true if all positive body atoms are true and no negative body atom is true.

The formalism can be understood more clearly by considering the following sentence as a simple example:

Computers are normally fast machines unless they are old.

This would be represented as a logical rule as follows:

$$\text{fastmachine}(X) \leftarrow \text{computer}(X), \mathbf{not} \text{old}(X).$$

where, X is a variable, *fastmachine*, *computer*, and *old* are predicates, and $\text{old}(X)$ is a negated atom.

Syntax. Let \mathcal{C} and \mathcal{V} be mutually disjoint sets of *constants* and *variables*, which we denote with first letter in lower case and upper case, respectively. Constants are used for constant terms, predicate names, and names for uninterpreted functions. The set of *terms* \mathcal{T} is recursively defined, it is the smallest set containing $\mathbb{N} \cup \mathcal{C} \cup \mathcal{V}$ as well as tuples of form (t_1, \dots, t_n) and uninterpreted function terms of form $f(t_1, \dots, t_n)$ where $f \in \mathcal{C}$ and $t_1, \dots, t_n \in \mathcal{T}$. An *ordinary atom* is of the form $p(t_1, \dots, t_n)$, where $p \in \mathcal{C}$, $t_1, \dots, t_n \in \mathcal{T}$, and $n \geq 0$ is the *arity* of the atom. An *aggregate atom* is of the form $X = \#agg\{t : b_1, \dots, b_k\}$ with variable $X \in \mathcal{V}$, aggregation function $\#agg \in \{\#sum, \#count\}$, with $1 < k$, $t \in \mathcal{T}$ and b_1, \dots, b_k a sequence of atoms. A term or atom is *ground* if it contains no sub-terms that are variables.

A *rule* r is of the form $\alpha \leftarrow \beta_1, \dots, \beta_n, \mathbf{not} \beta_{n+1}, \dots, \mathbf{not} \beta_m$ where $m \geq 0$, α is an ordinary atom, β_j , $0 \leq j \leq m$ is an atom, and we let $B(r) = \{\beta_1, \dots, \beta_n, \mathbf{not} \beta_{n+1}, \dots, \mathbf{not} \beta_m\}$ and $H(r) = \{\alpha\}$. A *program* is a finite set P of rules. A rule r is a *fact* if $m = 0$.

Semantics. Semantics of an ASP program P are defined using its Herbrand Base HB_P and its ground instantiation $grnd(P)$. An aggregate literal in the body of a rule accumulates truth values from a set of atoms, e.g., $N = \#count\{A : p(A)\}$ is true wrt. an interpretation $I \subseteq HB_P$ iff the extension of $p/1$ in I has size N . Using the usual notion of satisfying a rule with respect to a given interpretation, the FLP-reduct (Faber et al., 2011) fP^I reduces a program P using an answer set candidate I : $fP^I = \{r \in grnd(P) \mid I \models B(r)\}$. Finally iff I is a minimal model of fP^I .

Syntactic Sugar. Anonymous variables of form ‘ $_$ ’ are replaced by new variable symbols. Choice constructions can occur instead of rule heads, they generate a set of candidate solutions if the rule body is satisfied; e.g., $1\{p(a);p(b)\} \leq 2$ in the rule head generates all solution candidates where at least 1 and at most 2 atoms of the set $\{p(a),p(b)\}$ are true (bounds can be omitted). If a term is given as $X..Y$, where $X, Y \in \mathbb{N}$, then the rule containing the term is instantiated with all values from $\{v \in \mathbb{N} \mid X \leq v \leq Y\}$. A rule with $\alpha = \beta_{n+1}$, and α not occurring elsewhere in the program, is a *constraint*. Constraints eliminate answer sets I where $I \models B(r) \setminus \{\mathbf{not} \beta_{n+1}\}$, and we omit α and $\mathbf{not} \beta_{n+1}$ for constraints.

We refer to the ASP-Core-2 standard (Calimeri et al., 2012) and to books about ASP (Baral, 2004; Gebser et al., 2012a; Gelfond and Kahl, 2014) for a more elaborate description regarding the syntax and semantics.

Adding more knowledge results in a change of a previous understanding, this is common in human reasoning. Classical First-Order Logic does not allow such non-monotonic reasoning, however, ASP was designed as a commonsense reasoning formalism: a program has zero or more answer sets as solutions, adding knowledge to the program can remove answer sets as well as produce new ones. Note that ASP semantics rule out self-founded truths in answer sets. We use the ASP formalism due to its flexibility and declarativity.

ASP has been applied to several problems related to Natural Language Processing, see for example Mitra and Baral (2016); Schüller (2013, 2014, 2016); Schwitter (2012); Sharma et al. (2015), an overview of applications of ASP in general can be found in Erdem et al. (2016).

2.2 Inductive Logic Programming

Processing natural language based on hand-crafted rules is impractical, because human language is constantly evolving, partially due to the human creativity of language use. An example of this was recently noticed on UK highways where they advised drivers, ‘Don’t Pokémon Go and drive’. Pokémon Go is being informally used here as a verb even though it was only introduced as a game a few weeks before the sign was put up. To produce robust systems, it is necessary to use statistical models of language. These models are often pure Machine Learning (ML) estimators without any rule components (Manning and

Schütze, 1999). ML methods work very well in practice, however, they usually do not provide a way for explaining why a certain prediction was made, because they represent the learned knowledge in big matrices of real numbers. Some popular classifiers used for processing natural language include Naive Bayes, Decision Trees, Neural Networks (NNs), and Support Vector Machines (SVMs) (Dumais et al., 1998).

In this work, we focus on an approach that combines rule-based methods and statistics and provides interpretable learned models: *Inductive Logic Programming* (ILP). ILP is differentiated from ML techniques by its use of an expressive representation language and its ability to make use of logically encoded background knowledge (Muggleton and De Raedt, 1994). An important advantage of ILP over ML techniques such as neural networks is, that a hypothesis can be made readable by translating it into a piece of English text. Furthermore, if annotated corpora of sufficient size are not available or too expensive to produce, deep learning or other data-intense techniques are not applicable. However, we can still learn successfully with ILP.

Formally, ILP takes as input a set of examples E , a set B of background knowledge rules, and a set of mode declarations M , also called mode bias. As output, ILP aims to produce a set of rules H called hypothesis which entails E with respect to B (Otero, 2001).

$$B \wedge H \models E$$

The search for H with respect to E and B is restricted by M , which defines a language that limits the shape of rules in the hypothesis candidates and therefore the complexity of potential hypotheses.

Example 1. Consider the following example ILP instance (M, E, B) (Ray, 2009).

$$M = \left\{ \begin{array}{l} \#modeh \text{ flies}(+bird). \\ \#modeb \text{ penguin}(+bird). \\ \#modeb \text{ not penguin}(+bird). \end{array} \right\} \quad (2.1)$$

$$E = \left\{ \begin{array}{l} \#example \text{ flies}(a). \\ \#example \text{ flies}(b). \\ \#example \text{ flies}(c). \\ \#example \text{ not flies}(d). \end{array} \right\} \quad (2.2)$$

$$B = \left\{ \begin{array}{l} \text{bird}(X) \leftarrow \text{penguin}(X). \\ \text{bird}(a). \\ \text{bird}(b). \\ \text{bird}(c). \\ \text{penguin}(d). \end{array} \right\} \quad (2.3)$$

Based on this, an ILP system would ideally find the following hypothesis.

$$H = \left\{ \text{flies}(X) \leftarrow \text{bird}(X), \text{ not } \text{penguin}(X). \right\} \quad (2.4)$$

2.2.1 eXtended Hybrid Abductive Inductive Learning

The *eXtended Hybrid Abductive Inductive Learning* system (XHAIL) is an ILP approach based on ASP that generalises techniques of language and search bias from Horn clauses to normal logic programs with full usage of Negation as Failure (NAF) (Ray, 2009). Like its predecessor system Hybrid Abductive Inductive Learning (HAIL) which operated on Horn clauses, XHAIL is based on Abductive Logic Programming (ALP) (Kakas et al., 1992). XHAIL finds a hypothesis using several steps. Initially the examples E plus background knowledge B are transformed into a theory of Abductive Logic Programming (Kakas et al., 1992). The *Abduction* part of XHAIL explains observations with respect to a prior theory, which yields the *Kernel Set*, Δ . Δ is a set of potential heads of rules given by M such that a maximum of examples E is satisfied together with B . An overview of the XHAIL system can be seen in Figure 2.1

Example 2 (continued). *Given (M, E, B) from Example 1, XHAIL uses B , E , and the head part of M , to generate the Kernel Set Δ by abduction.*

$$\Delta = \left\{ \begin{array}{l} \text{flies}(a) \\ \text{flies}(b) \\ \text{flies}(c) \end{array} \right\}$$

The *Deduction* part uses Δ and the body part of the mode bias M to generate a ground program K . K contains rules which define atoms in Δ as true based on B and E .

The *Generalisation* part replaces constant terms in K with variables according to the mode

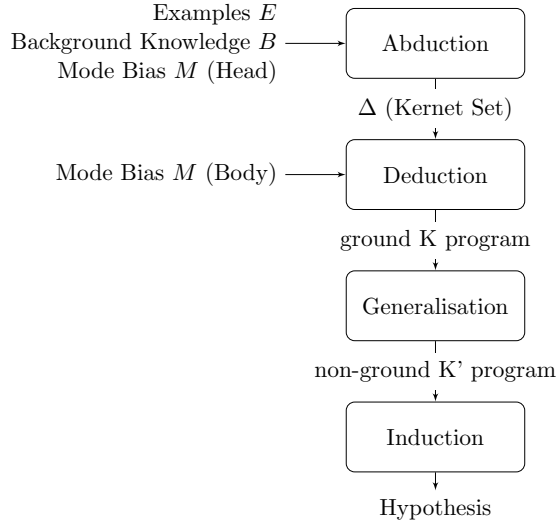


Figure 2.1: XHAIL architecture

bias M , which yields a non-ground program K' .

Example 3 (continued). *From the above Δ and M from (2.1), deduction and generalisation yield the following K and K' .*

$$K = \left\{ \begin{array}{l} \textit{flies}(a) \leftarrow \textit{bird}(a), \mathbf{not} \textit{penguin}(a) \\ \textit{flies}(b) \leftarrow \textit{bird}(b), \mathbf{not} \textit{penguin}(b) \\ \textit{flies}(c) \leftarrow \textit{bird}(c), \mathbf{not} \textit{penguin}(c) \end{array} \right\}$$

$$K' = \left\{ \begin{array}{l} \textit{flies}(X) \leftarrow \textit{bird}(X), \mathbf{not} \textit{penguin}(X) \\ \textit{flies}(Y) \leftarrow \textit{bird}(Y), \mathbf{not} \textit{penguin}(Y) \\ \textit{flies}(Z) \leftarrow \textit{bird}(Z), \mathbf{not} \textit{penguin}(Z) \end{array} \right\}$$

The *Induction* part searches for the smallest part of K' that entails as many examples of E as possible given B . This part of K' which can contain a subset of the rules of K' and for each rule, a subset of body atoms is called a hypothesis H .

Example 4 (continued). *The smallest hypothesis that covers all examples E in (2.2) is (2.4).*

We provide details of our work done with XHAIL in Chapter 3.3.

Chapter 3

Materials and Methods

In this chapter we mention the datasets used in our work. We also explain in detail each contribution made towards this Dissertation. We first discuss our system Inspire which predicts Interpretable Semantic Textual Similarity (iSTS) using hard-coded rules in ASP. Next we discuss how we extend the Inspire systems subtask of sentence chunking by using ILP in order to automate the task and learn the previously hard-coded rules. We also discuss how we modified the ILP-tool XHAIL in order to achieve the latter contribution.

3.1 Datasets

We are using the datasets from the SemEval2016 Task-2 iSTS (Agirre et al., 2016), which included two separate files containing sentence pairs. Three different datasets were provided: Headlines, Images, and Answers-Students. The Headlines dataset was mined from various news sources by European Media Monitor. The Images dataset is a collection of captions obtained from the Flickr dataset (Rashtchian et al., 2010). The Answers-Students corpus consists of the interactions between students and the BEETLE II tutorial dialogue system which is an intelligent tutoring engine that teaches students in basic electricity and electronics.

3.2 Predicting Interpretable Semantic Textual Similarity based on ASP

Semantic Textual Similarity (STS), refers to the degree of semantic equivalence between a pair of texts. This helps in explaining how some texts are related or unrelated. In Interpretable STS (iSTS) systems, further explanation is provided as to why the two texts are related or unrelated. Finding these detailed explanations helps in gathering a meaningful representation of their similarities.

The competition at SemEval 2016 for Task-2 (Agirre et al., 2016) was run on three different datasets: Headlines, Images and Student-Answers. Each dataset included two files containing pairs of sentences and two files containing pairs of gold-chunked sentences. Either the gold chunks provided by the organisers or chunks obtained from the given texts would be used as input to the system. The expected outputs of the system are m:n chunk-chunk alignments, corresponding similarity scores between 0 (unrelated) and 5 (equivalent), and a label indicating the type of semantic relation. All relations shall be considered in the given context. Possible semantic relations include:

- EQUI: Semantically equal
- OPPO: Opposite
- SPE1/SPE2: Chunk 1/2 is more specific than chunk 2/1
- SIMI: Similar but none of the relations above
- REL: Related but none of the relations above
- NOALI: Not aligned, e.g., punctuation
- FACT: Whether the chunk is a speculation or not
- POL: Whether the chunk is positive, negative or neutral

So for the sentence pair inputs:

Former Nazi death camp guard Demjanjuk dead at 91

John Demjanjuk, convicted Nazi death camp guard, dies aged 91

or for the chunked sentence pair inputs:

[Former Nazi death camp guard Demjanjuk] [dead] [at 91]
[John Demjanjuk] [,] [convicted Nazi death camp guard] [,] [dies] [aged 91]

we would obtain the following output for our iSTS system:

```
<sentence id="1" status="">
// Former Nazi death camp guard Demjanjuk dead at 91
// John Demjanjuk , convicted Nazi death camp guard , dies aged 91
<source>
1 Former :
2 Nazi :
3 death :
4 camp :
5 guard :
6 Demjanjuk :
7 dead :
8 at :
9 91 :
</source>
<translation>
1 John :
2 Demjanjuk :
3 , :
4 convicted :
5 Nazi :
6 death :
7 camp :
8 guard :
9 , :
10 dies :
11 aged :
12 91 :
</translation>
<alignment>
8 9 <=> 11 12 // EQUI // 5 // at 91 <=> aged 91
1 2 3 4 5 6 <=> 1 2 4 5 6 7 8 // SPE1_FACT // 3 // Former Nazi death camp
guard Demjanjuk <=> John Demjanjuk convicted Nazi death camp guard
```

```

7 <=> 10 // EQUI // 5 // dead <=> dies
0 <=> 3 // NOALI // 0 // -not aligned- <=> ,
0 <=> 9 // NOALI // 0 // -not aligned- <=> ,
</alignment>
</sentence>

```

3.2.1 Preprocessing

Facts that represent the input are created using lemmatization via NLTK (Bird, 2006), POS- and NER-tagging from Stanford CoreNLP (Manning et al., 2014), lookups in WordNet (Miller, 1995), and similarity values obtained using Word2Vec (Mikolov et al., 2013).

The following explains the input representation of the sentence pair:

```

[ A tan puppy ] [ being petted ] [ . ]
[ A tan puppy ] [ being held and petted ] [ . ]

```

Sentences, chunks, and words in chunks are first represented with POS-tags, NER-tags, and lowercase versions of words as follows:

```

sentence(1).
chunk(sc(1, 0)).
chunk(sc(1, 1)).
chunk(sc(1, 2)).
mword(cw(sc(1, 0), 1), "A", "a", "DT", "O").
mword(cw(sc(1, 0), 2), "tan",
      "tan", "NN", "O").
mword(cw(sc(1, 0), 3), "puppy",
      "puppy", "NN", "O").
...
sentence(2).
chunk(sc(2, 0)).
...

```

A function term $sc(sentenceID, chunkIdx)$ is built from an integer sentence ID, and a chunk ID, and a function term $cw(chunkID, wordIdx)$ is additionally built from a word ID.

Punctuation, cardinal numbers, and dates/times are detected using regular expressions and are represented as facts $punct(wordID)$, $cardinalnumber(wordID)$, and $datetime(wordID)$, resp., e.g.,

$$punct(cw(sc(1, 2), 0)).$$

Synonyms, hypernyms, and antonyms are looked up in WordNet and added as facts.

$$synonym("a", "a").$$

$$synonym("a", "vitamin a").$$

$$synonym("tan", "burn").$$

$$hypernym("puppy", "dog").$$

$$hypernym("puppy", "domestic_dog").$$

$$hypernym("puppy", "canis_familiaris").$$

...

Distributional similarity with the Word2Vec tool (Mikolov et al., 2013) is used to train on the One Billion Word¹ benchmark (Chelba et al., 2014) with SkipGram context representation, window size 10, vector dimension 200, and pruning below frequency 50. Word-word similarity $sim(v, w)$ is computed using cosine similarity from scikit-learn (Pedregosa et al., 2011), between vectors of words v and w . Chunk-chunk similarity is computed as

$$\frac{best(1, 2) + best(2, 1)}{2 \min(n_1, n_2)}, \text{ where} \tag{3.1}$$

$$best(x, y) = \sum_{i=1}^{n_x} \max_{j=1}^{n_y} sim(w_i^x, w_j^y)$$

with n_α the number of words in chunk α and w_β^α the word of index β in chunk α . This method for calculating chunk similarity is based on Banjade et al. (2015, Section 2.2.2).

¹<http://www.statmt.org/lm-benchmark/>

Chunk-chunk similarity is represented as atoms $chunksimilarity(chunk1Id, chunk2Id, S)$, where S , denotes the similarity score. In our example, this creates, e.g., the following facts.

$chunksimilarity(sc(1, 0), sc(2, 0), 101)$.

$chunksimilarity(sc(1, 0), sc(2, 1), 22)$.

$chunksimilarity(sc(1, 0), sc(2, 2), 2)$.

$chunksimilarity(sc(1, 1), sc(2, 0), 34)$.

...

Note that similarity can be above 100 due to dividing by the shorter chunk length.

3.2.2 Architecture

For each sentence pair, chunks are aligned according to the following architecture:

- chunked input sentence pairs are preprocessed (POS, NER, Word2Vec, WordNet) and represented as a set of ASP facts,
- a generic set of rules represents how alignments can be defined and changed,
- alignments are represented based on the description of the NeRoSim engine, and
- the above components are evaluated in an ASP solver, obtaining answer sets containing a representation of alignments which then generates the system output file.

3.2.2.1 Rule Engine

To make POS, NER, word, and lowercase words more accessible for manipulation, we project them to new facts with the following rules

$$\begin{aligned} \text{word}(Id, W) &\leftarrow \text{mword}(Id, W, -, -, -). \\ \text{lword}(Id, L) &\leftarrow \text{mword}(Id, -, L, -, -). \\ \text{pos}(Id, P) &\leftarrow \text{mword}(Id, -, -, P, -). \\ \text{ner}(Id, N) &\leftarrow \text{mword}(Id, -, -, -, N). \end{aligned}$$

where the arguments of *mword* are word ID, word, lowercase word, POS and NER tag.

Recall from Section 2.1 that variables of the form ‘_’ are anonymous, intuitively these values are projected away before applying the rule.

POS and NER tags, and mark nouns, verbs, contentwords, proper nouns, cardinal numbers and locations are interpreted based on tags from the Penn Treebank (Marcus et al., 1993).

$$\begin{aligned} \text{noun}(Id) &\leftarrow \text{pos}(Id, "NNS"). \\ \text{noun}(Id) &\leftarrow \text{pos}(Id, "NNP"). \\ \text{verb}(Id) &\leftarrow \text{pos}(Id, "VB"). \end{aligned}$$

$$\text{verb}(Id) \leftarrow \text{pos}(Id, "VBD").$$

...

$$\text{location}(Id) \leftarrow \text{ner}(Id, "LOCATION").$$

Symmetry of chunk similarity, synonyms and antonyms, and transitivity of the synonym relation is ensured.

$$\begin{aligned} \text{chunksimilarity}(C_2, C_1, S) &\leftarrow \text{chunksimilarity}(C_1, C_2, S). \\ \text{synonym}(W, W') &\leftarrow \text{synonym}(W', W). \\ \text{antonym}(W, W') &\leftarrow \text{antonym}(W', W). \\ \text{synonym}(W_1, W_3) &\leftarrow \text{synonym}(W_1, W_2), \text{synonym}(W_2, W_3). \end{aligned}$$

Pairs of chunks that can be matched together with their sentence indices are defined. This is useful because this way helps define conditions on potential alignments without specifying the direction of alignment (1 to 2 vs. 2 to 1).

$$chpair(S_1, S_2, sc(S_1, C_1), sc(S_2, C_2)) \leftarrow chunk(sc(S_1, C_1)), chunk(sc(S_2, C_2)), S_1 \neq S_2.$$

Alignments are represented as follows:

- (i) alignment happens in steps that have an order defined by atoms $nextStep(S, S')$ which indicates that S happens before S' ,
- (ii) a chunk can be aligned to one or more chunks only within one step, afterwards alignment cannot be changed,
- (iii) program modules can define atoms of form $chalign(C_1, R, Sc, C_2, St)$ which indicates that chunks C_1 and C_2 should be aligned with label R (e.g., "EQUI") and score Sc (e.g., 5) in step St .

Defining an alignment is possible if the chunks are not yet aligned, it marks both involved chunks as aligned, and they stay aligned.

$$aligned(C, S') \leftarrow \mathbf{not} aligned(C, S), chalign(C, -, -, -, S), nextStep(S, S').$$

$$aligned(C, S') \leftarrow \mathbf{not} aligned(C, S), chalign(-, -, -, C, S), nextStep(S, S').$$

$$aligned(C, S') \leftarrow aligned(C, S), nextStep(S, S').$$

Final alignments are represented using predicate $final$, these atoms include raw chunk similarity (for experimenting with other ways to define the similarity score). Moreover, only steps that are used (in $nextStep(\cdot, \cdot)$) are included.

$$usedStep(S) \leftarrow nextStep(S, -).$$

$$usedStep(S') \leftarrow nextStep(-, S').$$

$$final(C_1, Rel, Score, C_2, S, Simi) \leftarrow chalign(C_1, Rel, Score, C_2, S),$$

$$\mathbf{not} aligned(C_1, S), \mathbf{not} aligned(C_2, S),$$

$$usedStep(S), chunksimilarity(C_1, C_2, Simi).$$

This system gives us flexibility for configuring the usage and application of the order of rules by defining *nextStep* accordingly.

3.2.2.2 NeRoSim Rules

All that remains is to realise the NeRoSim rules, labeled with individual steps, and add them to the program.

In the following, we list how we realise NeRoSim’s conditions (Banjade et al., 2015, Section 3.1) that are checked before rule application:

- *condc₁*: “One chunk has a conjunction and other does not”

$$\begin{aligned} \text{condc}_1(C_1, C_2) \leftarrow \text{chpair}(-, -, C_1, C_2), \#count\{W_1 : \text{conjunction}(cw(C_1, W_1))\} = 0, \\ \#count\{W_2 : \text{conjunction}(cw(C_2, W_2))\} \geq 1. \end{aligned}$$

Intuitively, the *#count* aggregates become true if the appropriate number of atoms in the set becomes true.

- *condc₂*: “A content word in a chunk has an antonym in the other chunk”

$$\begin{aligned} \text{condc}_2(C_1, C_2) \leftarrow \text{chpair}(-, -, C_1, C_2), \text{contentword}(cw(C_1, WI_1)), \\ \text{lword}(cw(C_1, WI_1), W_1) \#count\{WI_2 : \text{lword}(cw(C_2, WI_2), W_2)\} \\ \text{antonym}(W_1, W_2). \end{aligned}$$

- *condc₃*: “A word in either chunk is a NUMERIC entity”

$$\text{condc}_3(C) \leftarrow \text{chunk}(C), \#count\{W : \text{cardinalnumber}(cw(C, W))\} \geq 1.$$

- *condc₄*: “Both chunks have LOCATION entities”

$$\text{condc}_4(C) \leftarrow \text{location}(cw(C, -)).$$

- $condc_5$: “Any of the chunks has a DATE/TIME entity”

$$condc_5(C) \leftarrow datetime(cw(C, -)).$$

- $condc_6$: “Both chunks share at least one content word other than noun”

$$\begin{aligned} condc_6(C_1, C_2) \leftarrow & match(cw(C_1, W_1), cw(C_2, W_2)), contentword(cw(C_1, W_1)), \\ & contentword(cw(C_2, W_2)), \mathbf{not} noun(cw(C_1, W_1)), \\ & \mathbf{not} noun(cw(C_2, W_2)). \end{aligned}$$

- $condc_7$: “Any of the chunks has a conjunction”

$$condc_7(C) \leftarrow chunk(C), conjunction(cw(C, -)).$$

We next detail the different NeRoSim rules (Banjade et al., 2015, Section 3.1.1-7) that we realise in ASP:

- no_1 : “If a chunk to be mapped is a single token and is a punctuation, assign NOALIC.”

$$no_1(C) \leftarrow chunk(C), pos(cw(C, WID), Pos), punct(Pos), len(C, w, 1).$$

- eq_1 : “Both chunks have same tokens (5)”

$$\begin{aligned} eq_1(C_1, C_2) \leftarrow & chpair(-, -, C_1, C_2), \mathbf{not} word_extra(C_1, C_2), \\ & \mathbf{not} word_extra(C_2, C_1). \end{aligned}$$

- eq_2 : “Both chunks have same content words (5)”

$$\begin{aligned} eq_2(C_1, C_2) \leftarrow & chpair(1, 2, C_1, C_2), contentword_match(C_1, C_2), \\ & \mathbf{not} contentword_extra(C_1, C_2), \mathbf{not} contentword_extra(C_2, C_1). \end{aligned}$$

- eq_3 : “All content words match using synonym lookup **(5)**”

$$eq_3(C_1, C_2) \leftarrow chpair(-, -, C_1, C_2), \mathbf{not} \ contentword_extra_notsynonym(C_1, C_2), \\ \mathbf{not} \ contentword_extra_notsynonym(C_2, C_1).$$

- eq_4 : “All content words of a chunk match and unmatched content word(s) of the other chunk are all of proper noun type **(5)**”

$$eq_4(C_1, C_2) \leftarrow chpair(-, -, C_1, C_2), \mathbf{not} \ cond1to5(C_1, C_2), \\ \mathbf{not} \ contentword_extra(C_1, C_2), \\ \mathbf{not} \ contentword_extra_notpropernoun(C_2, C_1).$$

- eq_5 : “Both chunks have equal number of content words and $sim\text{-}Mikolov(A, B) > 0.6$ **(5)**”

$$eq_5(C_1, C_2) \leftarrow eqcontentw(C_1, C_2), \mathbf{not} \ cond1235(C_1, C_2), \\ chunksimilarity(C_1, C_2, S), S > 60.$$

- op_1 : “A content word in a chunk has an antonym in the other chunk **(4)**”

$$op_1(C_1, C_2) \leftarrow chpair(-, -, C_1, C_2), condc_2(C_1, C_2), \\ \mathbf{not} \ cond3or7(C_1), \mathbf{not} \ cond3or7(C_2).$$

- sp_1 : “If chunk A but B has a conjunction and A contains all the content words of B then A is SPE of B **(4)**”

$$sp_1(C_1, C_2) \leftarrow chpair(-, -, C_1, C_2), condc_1(C_2, C_1), contentword_subset(C_2, C_1).$$

- sp_2 : “If chunk A contains all content words of chunk B plus some extra content words that are not verbs, A is a SPE of B or vice-versa. If chunk B has multiple SPEs, then

the chunk with the maximum token overlap with B is selected as the SPE of B. **(4)**”

$$\begin{aligned}
sp2candidate(A, B) \leftarrow & \text{chpair}(-, -, A, B), \text{contentword_subset}(B, A), \\
& \text{has_contentword_or_dontcare}(A), \\
& \text{has_contentword_or_dontcare}(B), \\
& 0 == \#count\{WId : \text{contentword_extra_w}(A, B, WId), \\
& \quad \text{verb}(WId).\} \\
sp2overlap(A, B, Overlap) \leftarrow & sp2candidate(A, B), \\
& \text{Overlap} = \#count\{WAIId : \text{match}(cw(A, WAIId), \\
& \quad cw(B, WBIId)).\} \\
\{sp2choose(A, B)\} \leftarrow & sp2bestoverlap(A, Highest), sp2overlap(A, B, Highest). \\
\leftarrow & sp2candidate(A, -), \mathbf{not} \ 1 = \#count\{A : sp2choose(A, B).\} \\
sp_2(A, B) \leftarrow & sp2choose(A, B).
\end{aligned}$$

- sp_3 : “If chunks A and B contain only one noun each say n_1 and n_2 and n_1 is hypernym of n_2 , B is SPE of A or vice versa **(4)**”

$$\begin{aligned}
sp_3(C_2, C_1) \leftarrow & sp3onenouneach(C_1, C_2), \text{noun}(cw(C_1, W_1)), \text{noun}(cw(C_2, W_2)), \\
& \text{lword}(cw(C_1, W_1), W1String), \text{lword}(cw(C_2, W_2), W2String), \\
& \text{hypernym}(W1String, W2String).
\end{aligned}$$

- si_1 : “Only the unmatched content word in each chunk is a CD type **(3)**”

$$\begin{aligned}
si_1(C_1, C_2) \leftarrow & si1candidate(C_1, C_2), \\
& \text{contentword_extra_w}(C_1, C_2, W_1), \text{cardinalnumber}(W_1), \\
& \text{contentword_extra_w}(C_2, C_1, W_2), \text{cardinalnumber}(W_2).
\end{aligned}$$

- si_2 : “Each chunk has a token of DATE/TIME type **(3)**”

$$si_2(C_1, C_2) \leftarrow \text{chpair}(1, 2, C_1, C_2), \text{condc}_5(C_1), \text{condc}_5(C_2).$$

- si_3 : “Each chunk has a token of LOCATION type **(3)**”

$$si_3(C_1, C_2) \leftarrow chpair(1, 2, C_1, C_2), condc_4(C_1), condc_4(C_2).$$

- si_4sim : “When both chunks share at least one noun then assign **3** if $sim\text{-Mikolov}(A, B) > 0.4$ and **2** otherwise.”

$$si_4sim(C_1, C_2, S) \leftarrow chpair(1, 2, C_1, C_2), match(cw(C_1, W_1), cw(C_2, W_2)), \\ noun(cw(C_1, W_1)), noun(cw(C_2, W_2)), \\ chunksimilarity(C_1, C_2, S).$$

- si_5sim : “This rule is applied only if C6 is not satisfied. Scores are assigned as : (i) **4** if $sim\text{Mikolov}(A, B) \in [0.7, 1.0]$ (ii) **3** if $sim\text{-Mikolov}(A, B) \in [0.65, 0.7)$ (iii) **2** if $sim\text{-Mikolov}(A, B) \in [0.60, 0.65)$ ”

$$si_5sim(C_1, C_2, S) \leftarrow chpair(1, 2, C_1, C_2), has_contentword_or_dontcare(C_1), \\ has_contentword_or_dontcare(C_2), \mathbf{not} condc_6(C_1, C_2), \\ chunksimilarity(C_1, C_2, S).$$

- re_1sim : “If both chunks share at least one content word other than noun then assign REL relation. Scores are assigned as follows : (i) **4** if $sim\text{-Mikolov}(A, B) \in [0.5, 1.0]$ (ii) **3** if $sim\text{-Mikolov}(A, B) \in [0.4, 0.5)$ (iii) **2** otherwise.”

$$re_1sim(C_1, C_2, S) \leftarrow chpair(1, 2, C_1, C_2), has_contentword_or_dontcare(C_1), \\ has_contentword_or_dontcare(C_2), condc_6(C_1, C_2), \\ chunksimilarity(C_1, C_2, S).$$

We define chunk alignments $chalign$ by using the above rules in our stepwise alignment engine in the following manner:

$$chalign(C_1, RULENAME, Score, C_2, rule) \leftarrow chpair(1, 2, C_1, C_2), rule(C_1, C_2).$$

where RULENAME can be NOALIC, EQUI, OPPO, SIMI, and REI paired with the corresponding rule to use for alignment (no_1, eq_1, etc). See Appendix A for the full ASP code.

3.2.2.3 Interpretation of Answer Sets

After the evaluation of the above rules with the facts that describe the input sentences the solver returns a set of answer sets (in our case a single answer set). This answer set contains all true atoms and we are interested only in the *final* predicates.

word(cw(sc(1, 1), 4), "being").

word(cw(sc(1, 1), 5), "petted").

word(cw(sc(2, 1), 4), "being").

word(cw(sc(2, 1), 5), "held").

word(cw(sc(2, 1), 6), "and").

...

final(sc(1, 0), "EQUI", 5, sc(2, 0), equi1, 101).

final(sc(1, 1), "SPE2", 4, sc(2, 1), sp1, 106).

Using Python and these predicates we create the required output which is a single continuous line of the following form:

```
4 5 6 <==> 4 5 6 7 // SPE2 // 4 // being petted <==> being held and petted
```

the above shows, the particular word tokens in a chunk from sentence 1 that are aligned to the corresponding word tokens from a chunk in sentence 2, the label given for similarity which in this case is specific to sentence 2, the similarity score and the actual chunks alignments of sentence 1 and 2.

3.2.2.4 Chunking based on ASP

For the sentence chunking subtask, the system has to identify chunks and align them. The Inspire system realises chunking as a preprocessing step: sentences are tokenised and

```

% prepositions are chunk starters
infrontof(X) :- form(X,"in"), form(X+1,"front"), form(X+2,"of").
% ignore the "of" if it is part of "in front of"
chunk(pp,X) :- pos(X,"IN"), not infrontof(X-2).
chunk(pp,X) :- form(X,"'s").
chunk(pp,X) :- pos(X,"TO").

% determiners are chunk starters, unless after preposition
chunk(dp,X) :- pos(X,"DT"), not pos(X-1,"IN").

% see annotation guidelines (Abney 1991),
% extended to include everything that is not a new chunk

% PPs have dependencies that the head depends on the %preposition (= start)
chunkAtLeastUntil(Start,Token) :- chunk(pp,Start), head(Token,Start).
% DPs have dependencies that the DP (= start) depends on the head
chunkAtLeastUntil(Start,Token) :- chunk(dp,Start), head(Start,Token).
% chunks extend to child tokens until the next chunk starts
chunkAtLeastUntil(Start,Token) :- chunkAtLeastUntil(Start,Until),
head(Token,Until), Until < Token,
0 = #count { T : chunk(_,T), Until <= T, T <= Token }.
% end of the chunk is the rightmost token
endOfChunk(Type,Start,End) :- chunk(Type,Start), token(End),
End = #max { Until : chunkAtLeastUntil(Start,Until) }.

% punctuations are chunk starters and enders
chunk(pt,X) :- pos(X,".").
chunk(pt,X) :- pos(X,",").
chunk(pt,X) :- pos(X,":").
chunk(pt,X) :- pos(X,";").
chunk(pt,X) :- pos(X,"'").
chunk(pt,X) :- pos(X,"'").
chunk(pt,X) :- pos(X,"'").
chunk(pt,X) :- pos(X,"\\").
% and so are VBZ/VBP (mostly)
chunk(pt,X) :- pos(X,"VBZ").
chunk(pt,X) :- pos(X,"VBP").
endOfChunk(pt,X,X) :- chunk(pt,X).

% certain relations start a chunk
chunk(preverb,X) :- head(X,Parent), rel(X,"APPO").
chunk(preverb,X) :- head(Parent,X), rel(Parent,"SBJ").

% adverbs start chunks
chunk(adv,X) :- pos(X,"RB").

chunk(X) :- chunk(_,X).
endOfChunk(C,X) :- endOfChunk(_,C,X).

% split between token X and X+1
split(X) :- token(X), chunk(X+1).
split(X) :- endOfChunk(_,X), token(X+1).

```

Figure 3.1: Manual Rules created for Sentence Chunking

processed by a joint POS-tagger and parser (Bohnet et al., 2013). Tokens, POS-tags, and dependency relations are represented as ASP facts and processed by a program that roughly encodes the following:

- chunks extend to child tokens until another chunk starts, and
- chunks start at
 - (i) prepositions, except ‘of’ in ‘in front of’;
 - (ii) determiners, unless after a preposition;
 - (iii) punctuations (where they immediately end);
 - (iv) adverbs;
 - (v) nodes in an appositive relation; and
 - (vi) nodes having a subject.

These rules were hard-coded by us to obtain a result close to Abney (1991). These can be seen in detail in Figure 3.1 which mentions where a chunk begins and ends. The presence of a chunk results in sentence splitting.

3.3 Chunking based on ILP with ASP

We extend the subtask of chunking in the Inspire system that was discussed in Section 3.2 by using ILP to learn the rules that were previously manually encoded for sentence chunking as mentioned in Section 3.2.2.4. We also introduce a best-effort strategy to extend the XHAIL system mentioned in Section 2.2.1 in order to make it computationally more efficient to use as our ILP solver. We then learn a rule based ASP program based on the knowledge and constraints provided by us; our ASP solver then utilises these rules in order to chunk the sentences.

Chunking (Tjong Kim Sang and Buchholz, 2000) or shallow parsing is the identification of short phrases such as noun phrases or prepositional phrases, usually based heavily on Part of Speech (POS) tags. POS provides only information about the token type, i.e., whether

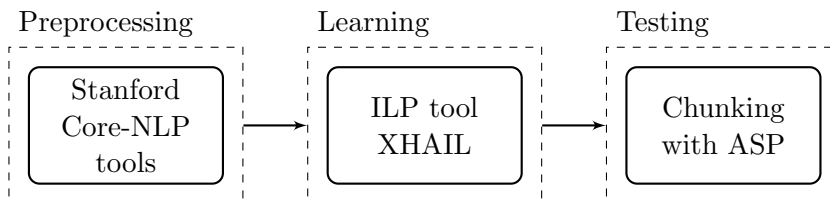


Figure 3.2: General overview of our framework

words are nouns, verbs, adjectives, etc., and chunking derives from that a shallow phrase structure, in our case a single level of chunks.

Our framework for chunking has three main parts as shown in Figure 3.2. Preprocessing is done using the Stanford CoreNLP tool from which we obtain the facts that are added to the background knowledge of XHAIL or used with a hypothesis to predict the chunks of an input. Using XHAIL as our ILP solver we learn a hypothesis (an ASP program) from the background knowledge, mode bias, and from examples which are generated using the gold-standard data. We predict chunks using our learned hypothesis and facts from preprocessing, using the Clingo (Gebser et al., 2008) ASP solver. We test by scoring predictions against gold chunk annotations.

Example 5. *An example sentence in the SemEval iSTS dataset (Agirre et al., 2016) is as follows.*

$$\textit{Former Nazi death camp guard Demjanjuk dead at 91} \quad (3.2)$$

The chunking present in the SemEval gold standard is as follows.

$$[\textit{Former Nazi death camp guard Demjanjuk}] [\textit{dead}] [\textit{at 91}] \quad (3.3)$$

3.3.1 Preprocessing

Stanford CoreNLP tools (Manning et al., 2014) are used for tokenisations and POS-tagging of the input. Using a shallow parser (Bohnet et al., 2013) we obtain the dependency relations for the sentences.

Our ASP representation contains atoms of the following form:

```

pos(c_NNP,1). head(2,1). form(1,"Former"). rel(c_NAME,1).
pos(c_NNP,2). head(5,2). form(2,"Nazi"). rel(c_NMOD,2).
pos(c_NN,3). head(4,3). form(3,"death"). rel(c_NMOD,3).
pos(c_NN,4). head(5,4). form(4,"camp"). rel(c_NMOD,4).
pos(c_NN,5). head(7,5). form(5,"guard"). rel(c_SBJ,5).
pos(c_NNP,6). head(5,6). form(6,"Demjanjuk"). rel(c_APPO,6).
pos(c_VBD,7). head(root,7). form(7,"dead"). rel(c_ROOT,7).
pos(c_IN,8). head(7,8). form(8,"at"). rel(c_ADV,8).
pos(c_CD,9). head(8,9). form(9,"91"). rel(c_PMOD,9).

```

(a) Preprocessing Output

```

postype(X) :- pos(X,-).
token(X) :- pos(-,X).
nextpos(P,X) :- pos(P,X+1).

```

(b) Background Knowledge

```

#modeh split(+token).
#modeb pos($postype,+token).
#modeb nextpos($postype,+token).

```

(c) Mode Restrictions

```

goodchunk(1) :- not split(1), not split(2), not split(3),
                not split(4), not split(5), split(6).
goodchunk(7) :- split(6), split(7).
goodchunk(8) :- split(7), not split(8).
#example goodchunk(1).
#example goodchunk(7).
#example goodchunk(8).

```

(d) Examples

Figure 3.3: XHAIL input for the sentence 'Former Nazi death camp guard Demjanjuk dead at 91' from the Headlines Dataset

- $pos(P, T)$ which represents that token T has POS tag P ,
- $form(T, Text)$ which represents that token T has surface form $Text$,
- $head(T_1, T_2)$ and $rel(R, T)$ which represent that token T_2 depends on token T_1 with dependency relation R .

Example 6. Figure 3.3a shows the result of preprocessing performed on sentence (3.2), which is a set of ASP facts.

We use Penn Treebank POS-tags as they are provided by Stanford CoreNLP. To form valid ASP constant terms from POS-tags, we prefix them with 'c_', replace special characters with lowercase letters (e.g., 'PRP\$' becomes 'c_PRPd'). In addition we create specific POS-tags for punctuation (see Section 5).

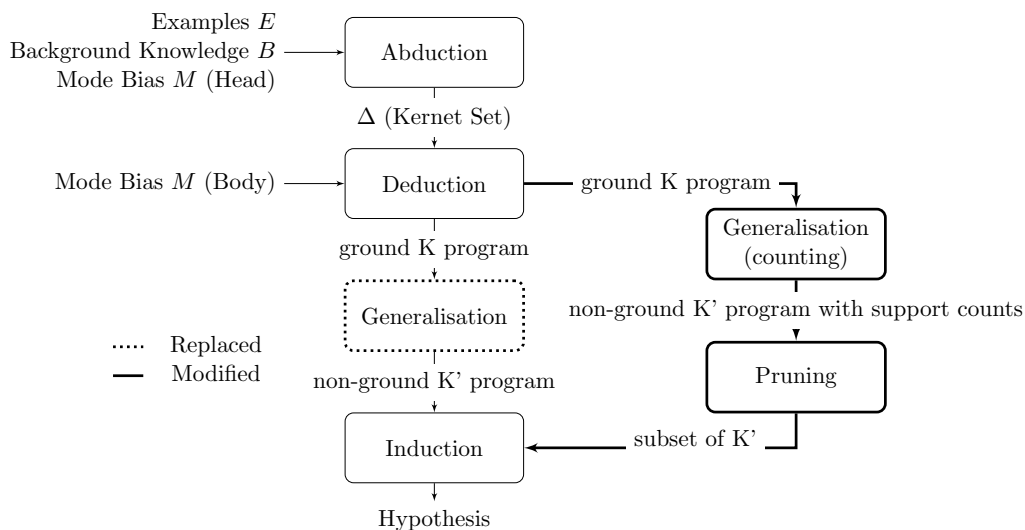


Figure 3.4: XHAIL architecture. The dotted line shows the replaced module with our version represented by the thick solid line.

3.3.2 Extension of XHAIL

Initially we intended to use the state-of-the-art ILP systems (ILASP2 or ILED) in our work . However, preliminary experiments with ILASP2 showed a lack in scalability (memory usage) even for only 100 sentences due to the unguided hypothesis search space. Moreover, experiments with ILED uncovered several problematic corner cases in the ILED algorithm that led to empty hypotheses when processing examples that were mutually inconsistent (which cannot be avoided in real-life NLP data). While trying to fix these problems in the algorithm, further issues in the ILED implementation came up. After consulting the authors of (Mitra and Baral, 2016) we learned that they had the same issues and used XHAIL, therefore we also opted to base our research on XHAIL due to it being the most robust tool for our task in comparison to the others.

Although XHAIL is applicable, we discovered several drawbacks and improved the approach and the XHAIL system. We provide an overview of the parts we changed, and then present our modifications. Figure 3.4 shows in the middle the original XHAIL components and on the right our extension.

We next describe our modifications of XHAIL.

3.3.2.1 Kernel Pruning according to Support

The computationally most expensive part of the search in XHAIL is Induction. Each non-ground rule in K' is rewritten into a combination of several guesses, one guess for the rule and one additional guess for each body atom in the rule.

We moreover observed, that some non-ground rules in K' are generalisations of many different ground rules in K , while some non-ground rules correspond with only a single instance in K . In the following, we say that the *support of r in K* is the number of ground rules in K that are transformed into $r \in K'$ in the Generalisation module of XHAIL (see Figure 3.4).

Intuitively, the higher the support, the more examples can be covered with that rule, and the more likely that rule or a part of it will be included in the optimal hypothesis.

Therefore we modified the XHAIL algorithm as follows.

- During Generalisation we keep track of the support of each rule $r \in K'$ by counting how often a generalisation yields the same rule r .
- We add an integer pruning parameter Pr to the algorithm and use only those rules from K' in the Induction component that have a support higher than Pr .

This modification is depicted as bold components which replace the dotted Generalisation module in Figure 3.4.

Pruning has several consequences. From a theoretical point of view, the algorithm becomes incomplete for $Pr > 0$, because Induction searches in a subset of the relevant hypotheses. Hence Induction might not be able to find a hypothesis that covers all examples, although such a hypothesis might exist with $Pr = 0$. From a practical point of view, pruning realises something akin to regularisation in classical ML; only strong patterns in the data will find their way into Induction and have the possibility to be represented in the hypothesis. A bit of pruning will therefore automatically prevent overfitting and generate more general hypotheses. As we will show in Experiments in Section 4, the pruning allows to configure a trade-off between considering low-support rules instead of omitting them entirely, as well as finding a more optimal hypothesis in comparison to a highly suboptimal one.

3.3.2.2 Unsat-core based and Best-effort Optimisation

We observed that ASP search in XHAIL Abduction and Induction components progresses very slowly from a suboptimal to an optimal solution. XHAIL integrates version 3 of Gringo (Gebser et al., 2011) and Clasp (Gebser et al., 2012b) which are both quite outdated. In particular Clasp in this version does not support three important improvements that have been developed by the ASP community for optimisation:

- (i) unsat-core optimisation (Andres et al., 2012),
- (ii) stratification for obtaining suboptimal models (Alviano et al., 2015b; Ansótegui et al., 2013), and
- (iii) unsat-core shrinking (Alviano and Dodaro, 2016).

Method (i) inverts the classical branch-and-bound search methodology which progresses from worst to better solutions. Unsat-core optimisation assumes all costs can be avoided and finds unsatisfiable cores of the problem until the assumption is true and a feasible solution is found. This has the disadvantage of providing only the final optimal solution, and to circumvent this disadvantage, stratification in method (ii) was developed which allows for combining branch-and-bound with method (i) to approach the optimal value both from cost 0 and from infinite cost. Furthermore, unsat-core shrinking in method (iii), also called ‘anytime ASP optimisation’, has the purpose of providing suboptimal solutions and aims to find smaller cores which can speed up the search significantly by cutting more of the search space (at the cost of searching for a smaller core). In experiments with the inductive encoding of XHAIL we found that all three methods have a beneficial effect.

Currently, only the WASP solver (Alviano et al., 2013, 2015a) supports all of (i), (ii), and (iii), therefore we integrated WASP into XHAIL, which has a different output format than Clasp. We also upgraded XHAIL to use Gringo version 4 which uses the new ASP-Core-2 standard and has some further (performance) advantages over older versions.

Unsat-core optimisation often finds solutions with a reasonable cost, near the optimal value, and then takes a long time to find the true optimum or prove optimality of the found solution. Therefore, we extended XHAIL as follows:

- a time budget for search can be specified on the command line,

- after the time budget is elapsed the best-known solution at that point is used and the algorithm continues, furthermore
- the distance from the optimal value is provided as output.

This affects the Induction step in Figure 3.4 and introduces a best-effort strategy; along with the obtained hypothesis we also get the distance from the optimal hypothesis, which is zero for optimal solutions.

Using a suboptimal hypothesis means, that either fewer examples are covered by the hypothesis than possible, or that the hypothesis is bigger than necessary. In practice, receiving a result is better than receiving no result at all, and our experiments show that XHAIL becomes applicable to reasonably-sized datasets using these extensions.

3.3.2.3 Other Improvements

We made two minor engineering contributions to XHAIL. A practically effective improvement of XHAIL concerns K' . As seen in Example 3, three rules that are equivalent modulo variable renaming are contained in K' . XHAIL contains canonicalization algorithms for avoiding such situations, based on hashing body elements of rules. However, we found that for cases with more than one variable and for cases with more than one body atom, these algorithms are not effective because XHAIL

- (i) uses a set data structure that maintains an order over elements,
- (ii) the set data structure is sensitive to insertion order, and
- (iii) hashing the set relies on the order to be canonical.

We made this canonicalization algorithm applicable to a far wider range of cases by changing the data type of rule bodies in XHAIL to a set that maintains an order depending on the value of set elements. This comes at a very low additional cost for set insertion and often reduces size of K' (and therefore computational effort for Induction step) without adversely changing the result of induction.

Another improvement concerns monitoring the ASP solving progress. The original implementation of XHAIL starts the external ASP solver and waits until the complete result is

received. During ASP solving, no output is processed, however ASP solvers provide output that is important for tracking the distance from optimality during a search. We extended XHAIL so that the output of the ASP solver can be made visible during the run using a command line option.

Chapter 4

Experiments

In this chapter we discuss the experiments carried out for iSTS and sentence chunking in ILP. For predicting the interpretable semantic textual similarity we set up three different runs based on optimising parameters for different criteria. For sentence chunking with ILP, the first step is to learn a model for each each sentence pair file present in each dataset obtained from the SemEval2016 Task-2 iSTS. The second step mentions the scoring that is used for our experiments, we have decided to use Precision, Recall and F1-score. Next we evaluate our results from the experiments for each dataset.

4.1 Predicting Interpretable Semantic Textual Similarity based on ASP

Our system does not require training, so we tested and optimised it on the training data for Headlines (H), Images (I), and Answers-Students (A-S) datasets. As criteria for accuracy, the competition used the F1 score based on alignments (Align), alignments and alignment type (Align+Type), alignments and alignment score (Align+Score), and full consideration of alignment, type, and score (Align+Type+Score).

Our optimisation experiments showed us, that there are significant differences in annotations between datasets. In particular, A-S contains spelling mistakes, verbs are often singleton chunks in H, and ‘to’ and ‘s’ often start a new chunk in H, while they are annotated as part of the previous chunk in I and A-S.

Therefore we decided to configure our system differently for each dataset, based on a Multinomial Naive Bayes Classifier trained on input unigrams and bigrams implemented using scikit-learn (Pedregosa et al., 2011). F1-score obtained on training data with 10-fold cross-validation was 0.99.

Our dataset configuration is as follows: we exclude stopwords from the calculation of similarity (3.1) for datasets H and I by using the NLTK corpus of stopwords; we remove non-singleton punctuation for dataset A-S; and we add rules to handle verb types (VBP, VBZ) as punctuation and ‘s’ as a preposition in chunking.

We optimised parameters for 3 runs according to different criteria and compared them to the Baseline (BL) provided by the organisers of the task.

4.1.1 Run 1

This run is optimised for the full label (Align+Type+Score). We used our implementation of NeRoSim rules in the same order, except SI4, SI5, and RE1 (Section 3.2.2.2), which we excluded. In ASP this is configured by defining facts for $nextStep(s, s')$ where

$$(s, s') \in \{(noalic, equi1), (equi1, equi2), (equi2, equi3), (equi3, equi4), \\ (equi4, equi5), (equi5, oppo), (oppo, sp1), (sp1, sp2), (sp2, sp3), \\ (sp3, simi1), (simi1, simi2), (simi2, simi3), (simi3, result)\}.$$

4.1.2 Run 2

This run is optimised for prediction of alignment (Align), this is done by using all NeRoSim rules in their original order: we define $nextStep(s, s')$ for

$$(s, s') \in \{(noalic, equi1), (equi1, equi2), (equi2, equi3), (equi3, equi4), (equi4, equi5), \\ (equi5, oppo), (oppo, sp1), (sp1, sp2), (sp2, sp3), (sp3, simi1), (simi1, simi2), \\ (simi2, simi3), (simi3, simi4), (simi4, simi5), (simi5, rel1), (rel1, result)\}.$$

In addition, for dataset A-S we perform automated spelling correction using Enchant.¹

¹<http://www.abisource.com/projects/enchant/>

4.1.3 Run 3

This run is based on the observation, that the scorer tool does not severely punish overlapping alignments in the F1-score of Align. Hence we allow SIMI4, SIMI5, and REL1 to be applied simultaneously by defining $nextStep(s, s')$ for

$$(s, s') \in \{(noalic, equi1), (equi1, equi2), (equi2, equi3), (equi3, equi4), (equi4, equi5), (equi5, oppo), (oppo, sp1), (sp1, sp2), (sp2, sp3), (sp3, simi1), (simi1, simi2), (simi2, simi3), (simi3, simi4), (simi3, simi5), (simi3, rel1), (simi4, result), (simi5, result), (rel1, result)\}.$$

Accordingly, we expected Run 1 to perform best with respect to the Align+Type+Score (and Align+Type) metric, Run 2 to perform best with respect to Align (and Align+Score) metrics, and Run 3 to sometimes perform above other runs. These expectations were confirmed by the results shown in the next section.

4.1.4 Scoring

The official evaluation (Melamed, 1998) uses the F1 of precision and recall of token alignments same as the case of Machine Translation. For each pair of chunks that are aligned, any pairs of tokens in the chunks are also aligned with some weight. The weight of each token-token alignment is the inverse of the number of alignments of each token (Agirre et al., 2016). Precision and recall are evaluated separately for all alignments of all pairs as follows:

$$Precision = \frac{TP}{SYS}$$
$$Recall = \frac{TP}{GOLD}$$

where TP is the number of system token-token alignments that are also present in the gold standard token-token alignments; SYS stands for the number of system alignments and GOLD stands for the number of gold standard alignments.

Participating runs were evaluated using four different metrics:

- F1 where alignment type and score are ignored
- F1 where alignment types need to match, but scores are ignored
- F1 where alignment type is ignored, but each alignment is penalised when scores do not match, and
- F1 where alignment types need to match, and each alignment is penalised when scores do not match.

The type and score F1 is the main overall metric. The evaluation procedure does not explicitly evaluate the chunking results.

4.2 Chunking based on ILP with ASP

In this section we explain in detail how we automate the previously discussed (See Section 3.2.2.4) subtask of sentence chunking in iSTS by learning the rules that were hand-crafted earlier in the Inspire system.

4.2.1 Model Learning

We are using the same datasets from the SemEval2016 Task-2 iSTS (Agirre et al., 2016), which included two separate files containing sentence pairs. In the following we denote S1 and S2, by sentence 1 and sentence 2 respectively, of sentence pairs in these datasets.

Regarding the size of the SemEval training dataset, Headlines and Images datasets are larger and contained 756 and 750 sentence pairs, respectively. However, the Answers-Students dataset was smaller and contained only 330 sentence pairs. In addition, all datasets contain a test portion of sentence pairs.

We use k -fold cross-validation to evaluate chunking with ILP, which yields k learned hypotheses and k evaluation scores for each parameter setting. We test each of these hypotheses also on the test portion of the respective dataset. From the scores obtained this way we compute mean and standard deviation, and perform statistical tests to find out whether observed score differences between parameter settings is statistically significant.

Dataset	Cross-Validation Set		Test Set	
	Size	Examples	S1	S2
H/I	100	S1 first 110	all	*
	500	S1 first 550	all	*
	100	S2 first 110	*	all
	500	S2 first 550	*	all
A-S	100	S1 first 55 + S2 first 55	all	all
	500	S1 first 275 + S2 first 275	all	all

Table 4.1: Dataset partitioning for 11-fold cross-validation experiments. Fields marked with * are not applicable, because we do not evaluate hypotheses learned from the S1 portion of the Headlines (H) and Images (I) datasets on the (independent) S2 portion of these datasets and vice versa. For the Answers-Students (A-S) dataset we need to merge S1 and S2 to obtain a model size of 500 from the training examples.

Table 4.1 shows which portions of the SemEval training dataset we used for 11-fold cross-validation. In the following, we call these datasets *Cross-Validation Sets*. We chose the first 110 and 550 examples to use for 11-fold cross-validation which results in training set sizes 100 and 500, respectively. As the Answers-Students dataset was smaller, we merged its sentence pairs in order to obtain a Cross-Validation Set size of 110 sentences, using the first 55 sentences from S1 and S2; and for 550 sentences, using the first 275 sentences from S1 and S2 each. As test portions we only use the original SemEval test datasets and we always test S1 and S2 separately.

Background Knowledge we use is shown in Figure 3.3b. We define which POS-tags can exist in predicate *postype/1* and which tokens exist in predicate *token/1*. Moreover, we provide for each token the POS-tag of its successors token in predicate *nextpos/2*.

Mode bias conditions are shown in Figure 3.3c, these limit the search space for hypothesis generation. Hypothesis rules contain as head atoms of the form, *split(T)*, which indicates, that a chunk ends at token T and a new chunk starts at token $T + 1$. The argument of predicates *split/1* in head is of type *token*.

The body of hypothesis rules can contain *pos/2* and *nextpos/2* predicates, where the first argument is a constant of type *postype* (which is defined in Figure 3.3b) and the second argument is a variable of type *token*. Hence this mode bias searches for rules defining chunk splits based on POS-tag of the token and the next token.

We deliberately use a very simple mode bias that does not make use of all atoms in the

facts obtained from preprocessing. This is discussed in Section 5.

4.2.2 Scoring

We use `diffib.SequenceMatcher` in Python to match the sentence chunks obtained from learning in ILP against the gold-standard sentence chunks. From the matchings obtained this way, we compute precision, recall, and F1-score as follows.

$$\begin{aligned} Precision &= \frac{\text{No. of Matched Sequences}}{\text{No. of ILP-learned Chunks}} \\ Recall &= \frac{\text{No. of Matched Sequences}}{\text{No. of Gold Chunks}} \\ Score &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned}$$

To investigate the effectivity of our mode bias for learning a hypothesis that can correctly classify the dataset, we perform cross-validation (see above) and measure correctness of all hypotheses obtained in cross-validation also on the test set.

Because of differences in S1/S2 portions of datasets, we report results separately for S1 and S2. We also evaluate classification separately for S1 and S2 for the Answers-Students dataset, although we train on a combination of S1 and S2.

4.2.3 Evaluation

We use Gringo version 4.5 (Gebser et al., 2011) and we use WASP version 2 (Git hash a44a95) (Alviano et al., 2015a) configured to use `unsat-core` optimisation with disjunctive core partitioning, core trimming, a budget of 30 seconds for computing the first model and for shrinking unsatisfiable cores with progressive shrinking strategy. These parameters were found most effective in preliminary experiments. We configure our modified XHAIL solver to allocate a budget of 1800 seconds for the Induction part which optimises the hypothesis (see Section 3.3.2.2). Memory usage never exceeded 5 GB.

Tables 4.2–4.4 contains the experimental results for each dataset, where columns *Size*, *Pr*, and *So*, respectively, show the number of sentences used to learn the model, the pruning

Size	Pr	So	S1						S2					
			CV			T			CV			T		
			F1	P	R	F1	R	F1	F1	P	R	F1	R	F1
100	0	172.8±46.2	66.3±10.1	63.0±2.2	64.9±3.3	59.4±3.3	70.7±14.2	65.5±2.4	64.6±2.7	60.5±2.6				
	1	10.9±5.0	71.1±13.3*	69.4±2.0	67.1±2.0	63.8±2.2*	69.3±15.7	67.3±0.5	66.2±1.4	62.4±1.0*				
	2	0.3±0.8	73.1±8.0	69.3±0.7	69.2±1.1	65.0±0.4*	66.5±15.4	65.9±1.5	68.2±0.5	62.7±1.1				
	3	0.0±0.0	65.9±5.9	66.6±3.4	69.7±1.7	63.0±2.9	65.1±15.6	64.7±0.9	68.5±0.3	61.6±0.5				
500	0	31954.4±7057.7	39.4±18.1	50.9±9.8	34.8±18.7	35.7±14.0	39.2±12.7	53.2±8.0	38.4±14.1	38.9±11.7				
	1	17855.1±6866.0	39.1±14.9	51.9±9.2	39.0±17.9	38.3±13.9	40.7±12.6	53.4±8.7	40.0±14.7	39.7±11.9				
	2	6238.5±1530.8	55.8±10.5*	59.6±4.2	57.0±9.2	53.2±6.8*	53.0±14.3*	59.4±5.7	52.0±11.9	49.4±9.5				
	3	4260.9±792.4	52.5±11.4	59.2±5.0	52.4±11.8	49.6±9.3*	59.2±7.8	62.1±2.9	58.5±4.6	54.7±3.5				
	4	1598.6±367.1	65.7±3.8*	65.2±3.3	66.3±3.0	61.1±3.0*	67.1±8.4*	63.3±2.0	64.7±4.1	59.7±3.0*				
	5	1117.2±211.3	67.0±4.6	66.8±3.1	67.8±3.1	62.9±3.0	73.3±7.3*	65.5±2.3	66.4±3.6	62.1±2.7*				
	6	732.6±130.4	69.7±4.0	67.5±1.9	69.5±2.4	64.3±2.1	71.7±5.7	65.3±1.6	67.4±4.4	62.7±2.9*				
	7	561.4±81.8	68.2±4.5	67.2±1.9	70.5±1.5	64.5±1.8	71.2±7.1	66.5±1.0	68.0±1.7	63.6±1.0				
	8	475.0±142.3	68.9±4.5	67.0±2.6	69.0±5.8	63.8±4.4	71.8±5.7	67.2±1.3	68.2±2.4	64.0±1.8				
	9	312.7±111.2	69.3±6.2	68.1±2.5	70.6±2.5	65.4±2.6	71.2±5.5	66.6±1.4	67.5±2.3	63.3±2.0				
10	220.3±59.9	67.8±4.5	67.3±2.1	70.9±2.8	65.0±2.4	73.4±6.7	66.1±1.7	68.6±2.1	63.7±1.6					

Table 4.2: Experimental Results for Headlines Dataset, where * indicates statistical significance ($p < 0.05$). Additionally, for Size = 500, the F1 scores for all pruning values $Pr > 1$ are significantly better than $Pr = 0$ ($p < 0.05$).

Size	Pr	So	S1						S2					
			CV			T			CV			T		
			F1	P	R	F1	R	F1	F1	P	R	F1	R	F1
100	0	0.5±1.0	81.8±12.7	66.4±15.5	74.3±0.7	73.7±0.7	73.9±0.7	60.1±9.5	60.1±9.5	60.1±9.6	60.1±9.5	60.1±9.5	60.1±9.5	
	1	0.0±0.0	80.9±14.4	64.5±10.8	72.7±1.4	72.1±1.4	72.3±1.4	50.2±5.6	50.2±5.6	50.0±5.6	50.1±5.6	50.1±5.6	50.1±5.6	
	2	0.0±0.0	78.2±15.3	64.5±13.7	69.2±1.4	68.9±0.8	68.9±1.2	47.5±1.8	47.5±1.8	47.3±1.8	47.4±1.8	47.4±1.8	47.4±1.8	
	3	0.0±0.0	72.7±14.2	66.4±16.7	67.0±0.5	67.8±0.5	67.1±0.5	47.3±1.5	47.3±1.5	47.1±1.5	47.2±1.5	47.2±1.5	47.2±1.5	
500	0	3797.3±1019.9	47.6±8.6	45.9±12.5	47.1±8.8	46.2±8.9	46.4±8.9	45.0±12.9	45.0±12.9	45.5±12.8	44.6±13.2	44.6±13.2	44.6±13.2	
	1	670.1±153.1	64.2±8.2	68.1±7.4	57.1±11.1	56.1±11.1	56.4±11.1	63.1±9.2	63.1±9.2	63.1±9.5	62.9±9.4	62.9±9.4	62.9±9.4	
	2	286.2±90.2	69.5±4.9	73.8±7.1	66.4±6.6	65.6±6.6	65.8±6.6	68.4±6.0	68.4±6.0	68.4±6.0	68.2±6.0	68.2±6.0	68.2±6.0	
	3	159.1±36.4	70.9±6.8	70.1±7.0	66.0±7.6	65.4±7.8	65.4±7.7	69.8±3.7	69.8±3.7	69.7±3.6	69.6±3.7	69.6±3.7	69.6±3.7	
	4	83.4±25.8	74.7±5.7	68.8±6.4	70.2±2.0	69.6±1.9	69.7±1.9	67.0±7.2	67.0±7.2	66.7±7.2	66.7±7.2	66.7±7.2	66.7±7.2	
	5	23.8±11.0	74.2±6.6	70.7±4.7	71.9±1.5	71.1±1.4	71.3±1.4	71.0±1.7	71.0±1.7	70.9±1.8	70.8±1.8	70.8±1.8	70.8±1.8	
	6	10.8±4.5	75.3±5.9	73.2±4.5	71.7±0.4	71.0±0.4	71.2±0.4	71.1±0.8	71.1±0.8	71.1±0.8	70.9±0.8	70.9±0.8	70.9±0.8	
	7	3.4±3.6	74.4±5.9	72.1±4.2	71.2±0.3	70.5±0.3	70.7±0.3	69.7±1.4	69.7±1.4	69.7±1.4	69.6±1.4	69.6±1.4	69.6±1.4	
	8	1.5±1.4	74.5±5.3	72.3±5.8	71.2±0.0	70.4±0.0	70.6±0.0	68.6±0.8	68.6±0.8	68.6±0.7	68.4±0.7	68.4±0.7	68.4±0.7	
	9	1.2±1.4	74.5±5.3	71.9±5.8	71.2±0.0	70.4±0.0	70.6±0.0	68.4±0.5	68.4±0.5	68.3±0.5	68.2±0.5	68.2±0.5	68.2±0.5	
	10	0.7±0.8	74.2±5.2	71.8±5.5	70.9±0.9	70.1±0.9	70.4±0.9	68.6±0.0	68.6±0.0	68.5±0.0	68.3±0.0	68.3±0.0	68.3±0.0	

Table 4.3: Experimental Results for Images Dataset, where * indicates statistical significance ($p < 0.05$). Additionally, for Size = 500, the F1 scores for all pruning values $Pr > 0$ are significantly better than $Pr = 0$ ($p < 0.05$).

Size	<i>Pr</i>	<i>So</i>	S1+S2			S1			S2		
			CV			T			T		
			F1	P	F1	R	F1	P	R	F1	P
100	0	93.2±22.6	66.1±12.9	69.3±1.5	63.2±3.2	61.0±2.6	89.3±3.0	80.1±0.7	80.3±1.7		
	1	5.3±4.6	67.0±11.5	67.9±1.3	61.6±1.8	59.4±1.7	87.7±1.0	79.7±0.8	79.5±1.0		
	2	0.0±0.0	65.0±10.8	67.7±0.7	64.9±1.4	61.2±0.8	86.3±1.5	80.2±0.5	78.4±1.4		
	3	0.0±0.0	64.8±10.4	66.8±0.4	63.0±1.4	59.9±0.5	86.6±1.5	80.7±0.3	78.9±1.4		
500	0	20723.5±3996.9	36.3±10.6	54.2±5.5	39.8±13.2	38.7±9.9	51.2±10.8	37.2±15.0	36.0±12.4	*	
	1	6643.4±1131.1	49.3±8.7	60.0±4.9	51.3±10.1	48.7±7.2	62.9±9.9	53.4±15.6	52.1±13.8	*	
	2	4422.2±734.7	54.5±9.7	62.6±2.4	60.6±8.1	56.1±5.5	66.4±7.0	59.9±11.7	57.6±10.8	*	
	3	2782.2±626.9	57.5±10.6	62.2±3.0	62.4±8.7	56.7±5.7	67.6±10.8	62.2±15.5	59.4±14.5	*	
	4	1541.6±311.3	65.5±4.1	66.8±2.8	70.5±2.5	63.5±2.4	78.9±2.0	79.5±2.0	76.0±2.0	*	
	5	1072.4±155.5	63.6±7.8	66.1±2.7	67.6±5.1	61.7±3.5	80.8±3.4	77.1±3.3	74.5±3.0		
	6	789.1±158.0	64.8±6.5	65.5±1.9	64.2±4.9	59.4±3.6	83.3±2.7	75.9±4.0	74.4±3.3		
	7	634.7±184.0	66.3±7.8	65.9±2.2	64.6±3.5	59.7±2.6	82.9±3.4	75.2±5.2	74.2±4.0		
	8	449.8±87.4	63.9±6.6	65.0±2.5	64.5±6.5	59.1±4.5	80.3±4.4	74.2±8.2	72.2±6.8		
	9	317.0±89.7	63.9±6.4	64.1±2.4	64.3±4.0	58.3±3.4	82.3±4.3	74.9±5.5	72.9±5.3		
	10	225.5±45.7	63.4±5.1	66.6±1.7	65.3±2.6	60.1±1.8	82.4±5.2	74.1±8.0	72.7±7.4		

Table 4.4: Experimental Results for Answers-Students Dataset, where * indicates statistical significance ($p < 0.05$). Additionally, for Size = 500, the F1 scores for all pruning values $Pr > 0$ are significantly better than $Pr = 0$ ($p < 0.05$).

parameter for generalising the learned hypothesis (see Section 3.3.2.1), and the rate of how close the learned hypothesis is to the optimal result, respectively. So is computed according to the following formula:

$$So = \frac{Upperbound - Lowerbound}{Lowerbound}$$

which is based on upper and lower bounds on the cost of the answer set. An So value of zero means optimality, and values above zero mean suboptimality; so the higher the value, the further away from optimality. Our results comprise of the mean and standard deviation of the F1-scores obtained from our 11-fold cross-validation test set of S1 and S2 individually (column CV). Due to lack of space, we opted to leave out the scores of precision and recall, but these values show similar trends as in the test set. For the test sets of both S1 and S2, we include the mean and standard deviation of the Precision, Recall and F1-scores (column group T).

When testing ML based systems, comparing results obtained on a single test set is often not sufficient, therefore we performed cross-validation to obtain mean and standard deviation about our benchmark metrics. This gives a better impression about the significance of the measured results. To obtain even more solid evidence, we additionally performed a one-tailed paired t-test to check if quality of results (e.g., F1 score) is significantly higher in one setting than in another one. We consider a result significant if $p < 0.05$, i.e., if there is a probability of less than 5 % that the result is due to chance. Our test is one-tailed because we check whether one result is higher than another one, and it is a paired test because we test different parameters on the same set of 11 training/test splits in cross-validation. There are even more powerful methods for proving significance of results such as bootstrap sampling (Efron and Tibshirani, 1986), however these methods require markedly higher computational effort in experiments and our experiments already show significance with the t-test.

Rows of Tables 4.2–4.4 contain results for learning from 100 resp. 500 example sentences, and for different pruning parameters. For each of the training set size, we increased pruning stepwise starting from value 0 until we found an optimal hypothesis ($So = 0$) or until we saw a clear peak in classification score in cross-validation (in that case, increasing the pruning is pointless, because it would increase optimality of the hypothesis but decrease the prediction scores).

Note that datasets have been tokenised very differently, and that also state-of-the-art systems in SemEval used separate preprocessing methods for each dataset. We follow this strategy to allow a fair comparison. One example for such a difference is the Images dataset, where the ‘.’ is considered as a separate token and is later defined as a separate chunk, however in Answers-Students dataset it is integrated into neighboring tokens.

Chapter 5

Results and Discussion

This chapter presents the results obtained by carrying out the experiments mentioned in the previous chapter. For iSTS we tabulate our results for both the subtasks and provide the scores obtained by the best system for each dataset in each subtask as well. For Chunking with ILP we provide a detailed discussion of the results obtained from the experimental evaluation and provide a comparison with the state-of-the-art systems in the given task.

5.1 Predicting Interpretable Semantic Textual Similarity based on ASP

The results of the competition, obtained with the above parameter sets, are shown in Table 5.1.

The Inspire system made use of a rule-based approach using Answer Set Programming for determining chunk boundaries (based on a representation obtained from a dependency parser) and for aligning chunks and assigning alignment type and score (based on a representation obtained from POS, NER, and distributed similarity tagging). In team ranking, our system is among the top three systems for Headlines and Images datasets, and in overall ranking (both for system and gold chunks). In terms of runs (each team could submit three runs), our system obtains first and second place for Headlines with gold standard chunks. For Answers-Students dataset our system performs worst. The configuration of Run 1 performs best in all categories.

We next work on extending the subtask of sentence chunking for the system Inspire by learning with ILP the ASP-based rules that are currently manually set.

Data	System	Align	Align+Type	Align+Score	Align+Type+Score
Gold-Standard Chunks					
H	BL	0.85	0.55	0.76	0.55
	r1	0.82	0.70	0.79	0.70
	r2	0.89	0.67	0.83	0.66
	r3	0.90	0.59	0.82	0.58
	Inspire_r1	0.82	0.70	0.79	0.70
I	BL	0.86	0.48	0.75	0.48
	r1	0.80	0.61	0.75	0.61
	r2	0.87	0.60	0.79	0.59
	r3	0.86	0.49	0.78	0.49
	UWB_r3	0.89	0.69	0.84	0.67
A-S	BL	0.82	0.56	0.75	0.56
	r1	0.80	0.51	0.73	0.51
	r2	0.82	0.48	0.74	0.48
	r3	0.87	0.39	0.77	0.39
	IISCNLP_r1	0.87	0.65	0.83	0.64
System Chunks					
H	BL	0.65	0.48	0.59	0.44
	r1	0.70	0.53	0.66	0.52
	r2	0.76	0.50	0.69	0.50
	r3	0.77	0.46	0.69	0.45
	DTSim_r2	0.84	0.56	0.76	0.55
I	BL	0.71	0.40	0.63	0.40
	r1	0.75	0.56	0.70	0.56
	r2	0.82	0.54	0.74	0.54
	r3	0.81	0.45	0.73	0.45
	DTSim_r3	0.84	0.63	0.78	0.61
A-S	BL	0.62	0.44	0.57	0.44
	r1	0.69	0.46	0.64	0.45
	r2	0.72	0.42	0.65	0.42
	r3	0.76	0.34	0.67	0.34
	FBK-HLT-NLP_r3	0.82	0.56	0.76	0.56

Table 5.1: System Performance results (F1-score) along with Best System Results for both subtasks and each dataset.

5.2 Chunking based on ILP with ASP

We first discuss the results of training set and pruning experiments, then compare our approach with the state-of-the-art systems, and finally inspect the optimal hypotheses.

5.2.1 Training Set and Pruning

Tables 4.2–4.4 show results of experiments, where T denotes the test portion of the respective dataset.

We observe that by increasing the size of the training set to learn the hypothesis, our scores improved considerably. Due to more information being provided, the learned hypothesis can predict with higher F1 score. We also observed that for the smaller training set size (100 sentences), lower pruning numbers (in rare cases even $Pr=0$) resulted in achieving the optimal solution. For a bigger training set size (500 sentences), without pruning the ILP procedure does not find solutions close to the optimal solution. However, by using pruning values up to $Pr=10$ we can reduce the size of the search space and find hypotheses closer to the optimum, which predict chunks with a higher F1 score. Our statistical test shows that, in many cases, several increments of the Pr parameter yield significantly better results, up to a point where prediction accuracy degrades because too many examples are pruned away.

To select the best hypothesis, we increase the pruning parameter Pr until we reach the peak in the F1 score in cross-validation.

Finding optimal hypotheses in the Inductive search of XHAIL (where $So=0$) is easily attained when learning from 100 sentences. For learning from 500 sentences, very high pruning results in a trivial optimal hypothesis (i.e., every token is a chunk) which has no predictive power, hence we do not increase Pr beyond a value of 10.

Note that we never encountered timeouts in the Abduction component of XHAIL, only in the Induction part. The original XHAIL tool without our improvements yields only timeouts for learning from 500 examples, and few hypotheses for learning from 100 examples. Therefore we do not show these results in tables.

5.2.2 State-of-the-art comparison

Table 5.2 shows a comparison of our results with the baseline and the three best systems from the chunking subtask of Task 2 from SemEval2016 Task2 (Agirre et al., 2016): DTSim (Banjade et al., 2016), FBK-HLT-NLP (Magnolini et al., 2016) and runs 1 and 2 of IISCNLP (Tekumalla and Jat, 2016). We also compare with results of our own system ‘Inspire-

Manual’ (Kazmi and Schüller, 2016).

- The baseline makes use of the automatic probabilistic chunker from the IXA-pipeline which provides Perceptron models (Collins, 2002) for chunking and is trained on CONLL2000 corpora and corrected manually,
- DTSim uses a Conditional Random Field (CRF) based chunking tool using only POS-tags as features,
- FBK-HLT-NLP obtains chunks using a Python implementation of MBSP chunker which uses a Memory-based part-of-speech tagger generator (Daelemans et al., 1996),
- Run 1 of IISCNLP uses OpenNLP chunker which divides the sentence into syntactically correlated parts of words, but does not specify their internal structure, nor their role in the main sentence. Run 2 uses Stanford NLP Parser to create parse trees and then uses a perl script to create chunks based on the parse trees, and
- Inspire-Manual (our previous system) makes use of manually set chunking rules (Abney, 1991) using ASP (Kazmi and Schüller, 2016).

Using the gold-standard chunks provided by the organisers we were able to compute the precision, recall, and F1-scores for analysis on the Headlines, Images and Answers-Students datasets.

For the scores of our system ‘Inspire-Learned’, we used the mean and average of the best configuration of our system as obtained in cross-validation experiments on the test set and compared against the other systems’ test set results. Our system’s performance is quite robust: it always scores within the top three best systems. Considering that we learn our hypothesis from a small portion of the training data based on only POS-tags, we conclude that our method is able to reach competitive results, with the additional bonus of providing an interpretable model.

5.2.3 Inspection of Hypotheses

Table 5.3 shows the rules that are obtained from the hypothesis generated by XHAIL from Sentence 1 files of all the datasets. We have also tabulated the common rules present

Dataset	System	S1			S2				
		P	R	F1	Rank	P	R	F1	Rank
Headlines	Baseline	60.5	36.6	37.6		63.6	42.5	42.8	
	DTSim	72.5	74.3	71.3	*	72.1	74.3	70.5	*
	FBK-HLT-NLP	63.6	51.3	51.5		57.1	51.1	48.3	
	IISCNLP - Run1	61.9	68.5	61.4		61.1	65.7	60.1	
	IISCNLP - Run2	67.6	68.5	64.5	***	71.4	71.9	68.9	**
	Inspire - Manual	64.5	70.4	62.4		64.3	68.4	62.2	
Inspire - Learned	68.1±2.5	70.6±2.5	65.4±2.6	**	67.2±1.3	68.2±2.4	64.0±1.8	***	
Images	Baseline	19.0	15.7	16.4		13.6	17.5	13.5	
	DTSim	77.8	77.4	77.5	*	79.5	79.1	79.2	*
	FBK-HLT-NLP	41.0	39.2	38.8		40.5	43.1	40.8	
	IISCNLP - Run1	61.6	60.9	60.7		66.1	66.2	65.9	
	IISCNLP - Run2	65.8	65.6	65.4		67.7	67.2	67.3	
	Inspire - Manual	74.5	74.2	74.2	**	73.8	73.6	73.6	**
Inspire - Learned	66.4±15.5	74.3±0.7	73.7±0.7	***	71.1±0.8	71.1±0.8	70.9±0.8	***	
Answers-Students	Baseline	62.1	30.9	34.6		59.2	33.4	36.6	
	DTSim	78.5	73.6	72.5	*	83.3	79.2	77.8	**
	FBK-HLT-NLP	70.3	52.5	52.8		72.4	59.1	59.3	
	IISCNLP - Run1	67.9	63.9	60.7	***	65.7	55.0	54.0	
	IISCNLP - Run2	63.0	59.8	56.9		66.2	52.5	52.8	
	Inspire - Manual	66.8	64.4	59.7		71.2	62.5	62.1	***
Inspire - Learned	66.8±2.8	70.5±2.5	63.5±2.4	**	89.3±3.0	80.1±0.7	80.3±1.7	*	

Table 5.2: Comparison with systems from SemEval 2016 Task 2. The number of stars shows the rank of the system.

Rules	H	I	A-S
split(V) :- token(V), pos(c_VBD,V).	X	X	X
split(V) :- token(V), nextpos(c_IN,V).	X	X	X
split(V) :- token(V), nextpos(c_VBZ,V).	X	X	X
split(V) :- token(V), pos(c_VB,V).	X		X
split(V) :- token(V), nextpos(c_TO,V).	X		X
split(V) :- token(V), nextpos(c_VBD,V).	X		X
split(V) :- token(V), nextpos(c_VBP,V).		X	X
split(V) :- token(V), pos(c_VBZ,V), nextpos(c_DT,V).		X	X
split(V) :- token(V), pos(c_NN,V), nextpos(c_RB,V).		X	X
split(V) :- token(V), pos(c_NNS,V).	X		
split(V) :- token(V), pos(c_VBP,V).	X		
split(V) :- token(V), pos(c_VBZ,V).	X		
split(V) :- token(V), pos(c_c,V).	X		
split(V) :- token(V), nextpos(c_POS,V).	X		
split(V) :- token(V), nextpos(c_VBN,V).	X		
split(V) :- token(V), nextpos(c_c,V).	X		
split(V) :- token(V), pos(c_PRP,V).		X	
split(V) :- token(V), pos(c_RP,V).		X	
split(V) :- token(V), pos(c_p,V).		X	
split(V) :- token(V), nextpos(c_p,V).		X	
split(V) :- token(V), pos(c_CC,V), nextpos(c_VBG,V).		X	
split(V) :- token(V), pos(c_NN,V), nextpos(c_VBD,V).		X	
split(V) :- token(V), pos(c_NN,V), nextpos(c_VBG,V).		X	
split(V) :- token(V), pos(c_NN,V), nextpos(c_VBN,V).		X	
split(V) :- token(V), pos(c_NNS,V), nextpos(c_VBG,V).		X	
split(V) :- token(V), pos(c_RB,V), nextpos(c_IN,V).		X	
split(V) :- token(V), pos(c_VBG,V), nextpos(c_DT,V).		X	
split(V) :- token(V), pos(c_VBG,V), nextpos(c_JJ,V).		X	
split(V) :- token(V), pos(c_VBG,V), nextpos(c_PRPd,V).		X	
split(V) :- token(V), pos(c_VBG,V), nextpos(c_RB,V).		X	
split(V) :- token(V), pos(c_VBZ,V), nextpos(c_IN,V).		X	
split(V) :- token(V), pos(c_EX,V).			X
split(V) :- token(V), pos(c_RB,V).			X
split(V) :- token(V), pos(c_VBG,V).			X
split(V) :- token(V), pos(c_WDT,V).			X
split(V) :- token(V), pos(c_WRB,V).			X
split(V) :- token(V), nextpos(c_EX,V).			X
split(V) :- token(V), nextpos(c_MD,V).			X
split(V) :- token(V), nextpos(c_VBG,V).			X
split(V) :- token(V), nextpos(c_RB,V).			X
split(V) :- token(V), pos(c_IN,V), nextpos(c_NNP,V).			X
split(V) :- token(V), pos(c_NN,V), nextpos(c_WDT,V).			X
split(V) :- token(V), pos(c_NN,V), nextpos(c_IN,V).			X
split(V) :- token(V), pos(c_NNS,V), nextpos(c_IN,V).			X
split(V) :- token(V), pos(c_NNS,V), nextpos(c_VBP,V).			X
split(V) :- token(V), pos(c_RB,V), nextpos(c_DT,V).			X

Table 5.3: Rules in the best hypotheses obtained from training on 500 sentences (S1), where X marks the presence of the rule in a given dataset.

between the datasets and the extra rules which differentiate the datasets from each other. POS-tags for punctuation are ‘c_p’ for sentence-final punctuation (‘.’, ‘?’, and ‘!’) and ‘c_c’ for sentence-separating punctuation (‘,’, ‘;’, and ‘:’).

Rules which occur in all learned hypotheses can be interpreted as follows:

- (i) chunks end at past tense verbs (VBD, e.g., ‘walked’),
- (ii) chunks begin at subordinating conjunctions and prepositions (IN, e.g., ‘in’), and
- (iii) chunks begin at 3rd person singular present tense verbs (VBZ, e.g., ‘walks’).

Rules that are common to H and A-S datasets are as follows:

- (i) chunks end at base forms of verbs (VB, e.g., ‘[to] walk’),
- (ii) chunks begin at ‘to’ prepositions (TO), and
- (iii) chunks begin at past tense verbs (VBD).

The absence of (i) in hypotheses for the Images dataset can be explained by the rareness of such verbs in captions of images. Note that (iii) together with the common rule (i) means that all VBD verbs become separate chunks in H and A-S datasets. Rules that are common to I and A-S datasets are as follows:

- (i) chunks begin at non-3rd person verbs in present tense (VBP, e.g., ‘[we] walk’),
- (ii) chunk boundaries are between a determiner (DT, e.g., ‘both’) and a 3rd person singular present tense verb (VBZ), and
- (iii) chunk boundaries are between adverbs (RB, e.g., ‘usually’) and common, singular, or mass nouns (NN, e.g., ‘humor’).

Interestingly, there are no rules common to H and I datasets except for the three rules mutual to all three datasets.

For rules occurring only in single datasets, we only discuss a few interesting cases in the following. Rules that are unique to the Headlines dataset include rules which indicate that

the sentence separators ‘,’ ‘;’, and ‘.’, become single chunks, moreover chunks start at genitive markers (POS, ‘s’). Both is not the case for the other two data sets. Rules unique to the Images dataset include that sentence-final punctuation (‘.’, ‘?’, and ‘!’) become separate chunks, rules for chunk boundaries between verb (VB_) and noun (NN_) tokens, and chunk boundaries between possessive pronouns (PRP\$, encoded as ‘c_PRPd’, e.g., ‘their’) and participles/gerunds (VBG, e.g., ‘falling’). Rules unique to Answers-Students dataset include chunks containing ‘existential there’ (EX), adverb tokens (RB), gerunds (VBG), and several rules for splits related to WH-determiners (WDT, e.g., ‘which’), WH-adverbs (WRB, e.g., ‘how’), and prepositions (IN). We see that our model is interpretable, which is not the case in classical ML techniques such as Neural Networks (NN), Conditional Random Fields (CRF), and Support Vector Machines (SVM).

5.2.4 Impact and Applicability

ILP is applicable to many problems of traditional ML, but usually only applicable for small datasets. Our addition of pruning enables learning from larger datasets at the cost of obtaining a more coarse-grained hypothesis and potentially suboptimal solutions.

The main advantage of ILP is interpretability and that it can achieve good results already with small datasets. Interpretability of the learned rule-based hypothesis makes the learned hypothesis transparent as opposed to black-box models of other approaches in the field such as Conditional Random Fields, Neural Networks, or Support Vector Machines. These approaches are often purely statistical, operate on big matrices of real numbers instead of logical rules, and are not interpretable. The disadvantage of ILP is that it often does not achieve the predictive performance of purely statistical approaches because the complexity of ILP learning limits the number of distinct features that can be used simultaneously.

Our approach allows finding suboptimal hypotheses which yield a higher prediction accuracy than an optimal hypothesis trained on a smaller training set. Learning a better model from a larger dataset is exactly what we would expect in ML. Before our improvement of XHAIL, obtaining any hypothesis from larger datasets was impossible: the original XHAIL tool does not return any hypothesis within several hours when learning from 500 examples.

Our chunking approach learns from a small portion of the full SemEval training dataset, based on only POS-tags, but it still achieves results close to the state-of-the-art. Addition-

ally it provides an interpretable model that allowed us to pinpoint non-uniform annotation practices in the three datasets of the SemEval 2016 iSTS competition. These observations give direct evidence for differences in annotation practice for three datasets with respect to punctuation and genitives, as well as differences in the content of the datasets.

5.2.5 Strengths and weaknesses

Our additions of pruning and the usage of suboptimal answer sets make ILP more robust because it permits learning from larger datasets and obtaining (potentially suboptimal) solutions faster.

Our addition of a time budget and usage of suboptimal answer sets is a purely beneficial addition to the XHAIL approach. If we disregard the additional benefits of pruning, i.e., if we disable pruning by setting $Pr=0$, then within the same time budget, the same optimal solutions are to be found as if using the original XHAIL approach. In addition, before finding the optimal solution, suboptimal hypotheses are provided in an online manner, together with information about their distance from the optimal solution.

The strength of pruning before the Induction phase is, that it permits learning from a bigger set of examples, while still considering all examples in the dataset. A weakness of pruning is, that a hypothesis which fits perfectly to the data might not be found anymore, even if the mode bias could permit such a perfect fit. In NLP applications this is not a big disadvantage, because noise usually prevents a perfect fit anyways, and overfitting models is indeed often a problem. However, in other application domains such as learning to interpret input data from user examples (Gulwani et al., 2015), a perfect fit to the input data might be desired and required. Note that pruning *examples* to learn from inconsistent data as done by Tang and Mooney (Tang and Mooney, 2001) is not necessary for our approach. Instead, non-covered examples incur a cost that is optimised to be as small as possible.

5.2.6 Design Decisions

In our study, we use a *simple mode bias* containing only the current and next POS tags, which is a deliberate choice to make results easier to compare. We performed experiments with additional body atoms *head/2* and *rel/2* in the body mode bias, moreover with

negation in the body mode bias. However, these experiments yielded significantly larger hypotheses with only small increases in accuracy. Therefore we here limit the analysis to the simple case and consider more complex mode biases as future work. Note that the best state-of-the-art system (DTSim) is a CRF model solely based on POS-tags, just as our hypothesis is only making use of POS-tags. By considering more than the current and immediately succeeding POS tag, DTSim can achieve better results than we do.

The *representation of examples* is an important part of our chunking case as described in Section 3.3. We define predicate *goodchunk* with rules that consider *presence* and *absence* of splits for each chunk. We make use of the power of NAF in these rules. We also experimented with an example representation that just gave all desired splits as `#example split(X)` and all undesired splits as `#example not split(Y)`. This representation contains an imbalance in the split versus `not split` class, moreover, chunks are not represented as a concept that can be optimised in the inductive search for the best hypothesis. Hence, it is not surprising that this simpler representation of examples gave drastically worse scores, and we do not report any of these results in detail.

Chapter 6

Related Work

6.1 Natural Language Processing with Inductive Logic Programming

From NLP point of view, the hope of ILP is to be able to steer a mid-course between these two alternatives of large-scale but shallow levels of analysis and small scale but deep and precise analysis. ILP should produce a better ratio between breadth of coverage and depth of analysis (Muggleton, 1999). ILP has been applied to the field of NLP successfully; it has not only been shown to have higher accuracies than various other ML approaches in learning the past tense of English but also shown to be capable of learning accurate grammars which translate sentences into deductive database queries (Law et al., 2014).

Except for one early application (Wirth, 1989) no application of ILP methods surfaced until the system CHILL (Mooney, 1996) was developed which learned a shift-reduce parser in Prolog from a training corpus of sentences paired with the desired parses by learning control rules and uses ILP to learn control strategies within this framework. This work also raised several issues regarding the capabilities and testing of ILP systems. CHILL was also used for parsing database queries to automate the construction of a natural language interface (Zelle and Mooney, 1996) and helped in demonstrating its ability to learn semantic mappings as well.

An extension of CHILL, CHILLIN (Zelle et al., 1994) was used along with an extension of FOIL, mFOIL (Tang and Mooney, 2001) for semantic parsing. Where CHILLIN com-

bines top-down and bottom-up induction methods and mFOIL is a top-down ILP algorithm designed keeping imperfect data in mind, which portrays whether a clause refinement is significant for the overall performance with the help of a pre-pruning algorithm. This emphasised on how the combination of multiple clause constructors helps improve the overall learning; which is a rather similar concept to Ensemble Methods in standard ML. Note that CHILLIN pruning is based on probability estimates and has the purpose of dealing with inconsistency in the data. Opposed to that, XHAIL already supports learning from inconsistent data, and the pruning we discuss in Section 3.3.2.1 aims to increase scalability.

Previous work ILP systems such as TILDE and Aleph (Srinivasan, 2001) have been applied to preference learning which addressed learning ratings such as good, poor and bad. ASP expresses preferences through weak constraints and may also contain weak constraints or optimisation statements which impose an ordering on the answer sets (Law et al., 2015).

The system of Mitra and Baral (2016) uses ASP as primary knowledge representation and reasoning language to address the task of Question Answering. They use a rule layer that is partially learned with XHAIL to connect results from an Abstract Meaning Representation parser and an Event Calculus theory as background knowledge.

6.2 Systems for Inductive Logic Programming

The following sections give an overview of ILP systems based on ASP that are designed to operate in the presence of negation, in addition to XHAIL that we introduced in detail in Section 2.2.1.

6.2.1 Inductive Learning of Answer Set Programs

The *Inductive Learning of Answer Set Programs* approach (ILASP) is an extension of the notion of learning from answer sets (Law et al., 2014). Importantly, it covers positive examples bravely (i.e., in at least one answer set) and ensures that the negation of negative examples is cautiously entailed (i.e., no negative example is covered in any answer set) (Otero, 2001). Negative examples are needed to learn Answer Set Programs with non-determinism otherwise there is no concept of what should *not* be in an Answer Set. ILASP conducts a

search in multiple stages for brave and cautious entailment and processes all examples at once. ILASP performs a less informed hypothesis search than XHAIL or ILED, that means large hypothesis spaces are infeasible for ILASP while they are not problematic for XHAIL and ILED, on the other hand, ILASP supports aggregates and constraints while the older systems do not support these.

ILASP2 (Law et al., 2015) extends the hypothesis space of ILASP with choice rules and weak constraints. This permits searching for hypotheses that encode preference relations. ILASP is more expressive than XHAIL but is less scalable, therefore we based our work on XHAIL.

6.2.2 Incremental Learning of Event Definitions

The *Incremental Learning of Event Definitions* (ILED) algorithm (Katzouris et al., 2015) relies on Abductive-Inductive learning and comprises of a scalable clause refinement methodology based on a compressive summarization of clause coverage in a stream of examples. Previous ILP learners were batch learners and required all training data to be in place prior to the initiation of the learning process. ILED learns incrementally by processing training instances when they become available and altering previously inferred knowledge to fit new observation, this is also known as theory revision. It exploits previous computations to speed-up the learning since revising the hypothesis is considered more efficient than learning from scratch. ILED attempts to cover a maximum of examples by re-iterating over previously seen examples when the hypothesis has been refined. While XHAIL can ensure optimal example coverage easily by processing all examples at once, ILED does not preserve this property due to a non-global view on examples.

When considering ASP-based ILP, negation in the body of rules is not the only interesting addition to the overall concept of ILP. An ASP program can have several independent solutions, called answer sets, of the program. Even the background knowledge B can admit several answer sets without any addition of facts from examples. Therefore, a hypothesis H can cover some examples in one answer set, while others are covered by another answer set. XHAIL and ILED approaches are based on finding a hypothesis that is covering all examples in a *single* answer set.

Unfortunately, in our experiments, we encountered several problems with ILED in the pres-

ence of inconsistent input data (which is typical in NLP). Therefore, we decided to use XHAIL as our ILP tool.

6.2.3 Inspire-ILP

The Inspire system (Schüller, 2017) is an Inductive Logic Programming system based on an ASP encoding for generating hypotheses with a cost from the mode bias, and a transformation of hypotheses and examples to an ASP optimisation problem that has the smallest hypothesis covering all examples as solutions. Inspire attempts to learn a hypothesis on single examples while increasing maximum hypothesis cost.

The approach has the advantage that there is no need to make abduction of required facts, then induction of potential rules, then generalisation of these rules, then a search for the smallest hypothesis (as done in the XHAIL (Ray, 2009) system) while the obvious disadvantage is, that hypothesis search is blind (similar as in the ILASP (Law et al., 2014) system).

For our application, learning from a single example is not useful as providing more examples gives more information to the ILP tool. Hence, we use XHAIL in order to use all examples at once in order to learn a better and stronger hypothesis for our work.

Chapter 7

Conclusion and Future Work

The Inspire system made use of a rule-based approach using Answer Set Programming for determining chunk boundaries based on a representation obtained from a dependency parser, and for aligning chunks and assigning alignment type and score based on a representation obtained from POS, NER, and distributed similarity tagging. Our system competed at the SemEval2016 Task-2 iSTS competition and was among the top three systems for Headlines and Images datasets, and in overall ranking both for system and gold chunks subtasks. In terms of runs (each team could submit three runs), our system obtains the first and second place for Headlines with gold standard chunks. For Student-Answers dataset our system performs worst. The configuration of Run 1 performs best in all categories.

In the subtask of the competition for chunking, we decided to extend our system by learning our previous technique of rule-based chunking. We explore the usage of ILP for the NLP task of chunking. ILP combines logic programming and Machine Learning (ML), and it provides interpretable models, i.e., logical hypotheses, which are learned from data. ILP has been applied to a variety of NLP and other problems such as parsing (Tang and Mooney, 2001; Zelle and Mooney, 1996), automatic construction of biological knowledge bases from scientific abstracts (Craven and Kumlien, 1999), automatic scientific discovery (King et al., 2004), and in Microsoft Excel Gulwani et al. (2015) where users can specify data extraction rules using examples. Therefore, ILP research has the potential for being used in a wide range of applications.

We extend the XHAIL ILP solver to increase its scalability and applicability for the task of sentence chunking and the results indicate that ILP is competitive to state-of-the-art ML

Feature	ILP	ML	Rules
Expressivity	X		X
Learning performance from less training data	X		
Interpretable Model	X		
Ability to track learned model	X	X (Decision Trees)	
Representation of examples shaping learned model	X		
Robustness to new input	X	X	
Low engineering complexity		X	

Table 7.1: Comparison of the benefits and drawbacks provided by Inductive Logic Programming over standard Machine Learning Techniques and manually-set rules

techniques which can be seen listed in Table 7.1. We have successfully extended XHAIL to allow learning from larger datasets than previously possible. Learning a hypothesis using ILP has the advantage of an interpretable representation of the learned knowledge, such that we know exactly which rule has been learned by the program and how it affects our NLP task. In this study, we also gain insights about the differences and common points of datasets that we learned a hypothesis from. Moreover, ILP permits learning from small training sets where techniques such as Neural Networks fail to provide good results.

As a first contribution to the ILP tool XHAIL we have upgraded the software so that it uses the newest solver technology, and that this technology is used in a best-effort manner that can utilise suboptimal search results. This is effective in practice, because finding the optimal solution can be disproportionately more difficult than finding a solution close to the optimum. Moreover, the ASP technique we use here provides a clear information about the degree of suboptimality. During our experiments, a new version of Clingo was published which contains most techniques in WASP (except for core shrinking). We decided to continue using WASP for this study because we saw that core shrinking is also beneficial to search. Extending XHAIL to use Clingo in a best-effort manner is quite straight-forward.

As a second contribution to XHAIL we have added a *pruning* parameter to the algorithm that allows fine-tuning the search space for hypotheses by filtering out rule candidates that are supported by fewer examples than other rules. This addition is a novel contribution to the algorithm, which leads to significant improvements in efficiency, and increases the number of hypotheses that are found in a given time budget. While pruning makes the method incomplete, it does not reduce expressivity. The hypotheses and background knowledge may still contain unrestricted Negation as Failure. Pruning in our work is similar to the concept of the regularisation in ML and is there to prevent overfitting in the hypothesis generation.

Pruning enables the learning of logical hypotheses with dataset sizes that were not feasible before. We experimentally observed a trade-off between finding an optimal hypothesis that considers all potential rules on one hand, and finding a suboptimal hypothesis that is based on rules that are supported by few examples. Therefore the pruning parameter has to be adjusted on an application-by-application basis.

Note that, in XHAIL, pruning examples to learn from inconsistent data as done by Tang and Mooney (2001) is not necessary. Instead, non-covered examples incur a cost that is optimised via ASP. Our additional pruning step enables learning from a bigger amount of examples in this setting. We provide the modified XHAIL in a public repository fork (Bragaglia and Schüller, 2016).

Our work has focused on providing comparable results to ML techniques and we have not utilised the full power of ILP with Negation as Failure (NAF) in rule bodies and predicate invention. As future work, we plan to extend the predicates usable in hypotheses to provide a more detailed representation of the NLP task, moreover we plan to enrich the background knowledge to aid ILP in learning a better hypothesis with a deeper structure representing the boundaries of chunks.

Bibliography

- Steven P Abney. Parsing by chunks. In *Principle-based parsing*, pages 257–278. 1991.
- Eneko Agirre, Aitor Gonzalez-Agirre, Iñigo Lopez-Gazpio, Montse Maritxalar, German Rigau, and Larraitz Uria. SemEval-2016 task 2: Interpretable Semantic Textual Similarity. In *Proceedings of SemEval*, pages 512–524, 2016.
- Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *Theory and Practice of Logic Programming*, 16(5-6):533–551, 2016.
- Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In *Logic Programming and Nonmonotonic Reasoning*, pages 54–66, 2013.
- Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *Logic Programming and Nonmonotonic Reasoning* , pages 40–54, 2015a.
- Mario Alviano, Carmine Dodaro, Joao Marques-Silva, and Francesco Ricca. Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*, page exv061, 2015b.
- Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *International Conference on Logic Programming, Technical Communications*, pages 212–221, 2012.
- Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.
- Rajendra Banjade, Nobal B Niraula, Nabin Maharjan, Vasile Rus, Dan Stefanescu, Mihai Lintean, and Dipesh Gautam. NeRoSim: A System for Measuring and Interpreting Semantic Textual Similarity. In *SemEval 2015*, pages 164–171, 2015.

- Rajendra Banjade, Nabin Maharjan, Nobal B Niraula, and Vasile Rus. DTSim at SemEval-2016 task 2: Interpreting Similarity of Texts Based on Automated Chunking, Chunk Alignment and Semantic Relation Prediction. In *Proceedings of SemEval*, pages 809–813, 2016.
- Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, 2004. ISBN 0511042760.
- Steven Bird. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72, 2006.
- Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428, 2013.
- Stefano Bragaglia and Peter Schüller. XHAIL (Version 4c5e0b8) [System for eXtended Hybrid Abductive Inductive Learning], 2016. Retrieved from <https://github.com/knowlp/XHAIL>.
- Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
- Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 Input language format. Technical report, ASP Standardization Working Group, 2012.
- Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. Technical report, 2014.
- William Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.
- Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

- Mark Craven and Johan Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 77–86, 1999.
- Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. Mbt: A memory-based part of speech tagger-generator. *arXiv preprint cmp-lg/9607012*, 1996.
- Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, pages 148–155, 1998.
- B Efron and R Tibshirani. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statistical Science*, 1(1):54–75, 1986.
- Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of Answer Set Programming. *AI Magazine*, 37(3):53–68, 2016.
- Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011. ISSN 00043702.
- M Gebser, R Kaminski, B Kaufmann, M Ostrowski, T Schaub, and S Thiele. A user’s guide to gringo, clasp, clingo, and iclingo. Technical report, University of Potsdam, 2008.
- Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in gringo series 3. In *International Conference on Logic Programming and Non-monotonic Reasoning*, pages 345–351, 2011. ISBN 9783642208942. doi: 10.1007/978-3-642-20895-9_39.
- Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012a.
- Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012b.
- Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

- Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.
- Sumit Gulwani, Jose Hernandez-Orallo, Emanuel Kitzelmann, Stephen H Muggleton, Ute Schmid, and Benjamin Zorn. Inductive programming meets the real world. *Communications of the ACM*, 58(11):90–99, 2015.
- A C Kakas, R A Kowalski, and F Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992. doi: 10.1093/logcom/2.6.719.
- Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.
- Mishal Kazmi and Peter Schüller. Inspire at SemEval-2016 task 2: Interpretable semantic textual similarity alignment based on answer set programming. In *Proceedings of SemEval*, pages 1109–1115, 2016.
- Ross D King, Kenneth E Whelan, Ffion M Jones, Philip G K Reiser, Christopher H Bryant, Stephen H Muggleton, Douglas B Kell, and Stephen G Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, 2004. doi: 10.1038/nature02236.
- Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *European Workshop on Logics in Artificial Intelligence*, pages 311–325, 2014.
- Mark Law, Alessandra Russo, and Krysia Broda. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):511–525, 2015.
- Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002. doi: 10.1016/s0004-3702(02)00186-8.
- Vladimir Lifschitz. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.
- John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.
- Simone Magnolini, Anna Feltracco, and Bernardo Magnini. FBK-HLT-NLP at SemEval-2016 task 2: A multitask, deep learning approach for interpretable semantic textual similarity. In *Proceedings of SemEval*, pages 783–789, 2016.

- Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing (Vol. 999)*. Cambridge:MIT Press, 1999.
- Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL System Demonstrations*, pages 55–60, 2014.
- Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.
- I Dan Melamed. Manual annotation of translational equivalence: The blinker project. *arXiv preprint cmp-lg/9805005*, 1998.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- Arindam Mitra and Chitta Baral. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *Association for the Advancement of Artificial Intelligence*, pages 2779–2785, 2016.
- Raymond J Mooney. Inductive logic programming for natural language processing. In *Inductive Logic Programming*, pages 1–22. 1996.
- Stephen Muggleton. Inductive logic programming: issues, results and the challenge of learning language in logic. *Artificial Intelligence*, 114(1-2):283–296, 1999.
- Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.
- Ramón P Otero. Induction of stable models. In *International Conference on Inductive Logic Programming*, pages 193–205. Springer, 2001.
- Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al.

- Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12 (Oct):2825–2830, 2011.
- Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. Collecting image annotations using Amazon’s Mechanical Turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk*, pages 139–147, 2010.
- Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3): 329–340, 2009.
- Peter Schüller. Flexible Combinatory Categorical Grammar Parsing using the CYK Algorithm and Answer Set Programming. In *International Conference on Logic Programming and Non-monotonic Reasoning*, pages 499–511, 2013.
- Peter Schüller. Tackling Winograd Schemas by Formalizing Relevance Theory in Knowledge Graphs. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 358–367. AAAI Press, 2014.
- Peter Schüller. Modeling Variations of First-Order Horn Abduction in Answer Set Programming. *Fundamenta Informaticae*, 149:159–207, 2016. doi: 10.3233/FI-2015-1446.
- Peter Schüller. Inspire at inductive logic programming competition: Fine-grained cost-based hypothesis generation. Technical report, 2017.
- Rolf Schwitter. Answer Set Programming via Controlled Natural Language Processing. In *Controlled Natural Language*, pages 26–43, 2012.
- Arpit Sharma, Nguyen H. Vo, Somak Aditya, and Chitta Baral. Towards addressing the winograd schema challenge - Building and using a semantic parser and a knowledge hunting module. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1319–1325, 2015. ISBN 9781577357384.
- Ashwin Srinivasan. The aleph manual, 2001. Retrieved from <http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph.html>.
- Lappoon R Tang and Raymond J Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pages 466–477, 2001.

- Lavanya Tekumalla and Sharmistha Jat. IISCNLP at SemEval-2016 task 2: Interpretable STS with ILP based Multiple Chunk Aligner. In *Proceedings of SemEval*, pages 790–795, 2016.
- Erik F Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Workshop on Learning Language in Logic and Conference on Computational Natural Language Learning*, pages 127–132, 2000.
- Ruediger Wirth. Completing logic programs by inverse resolution. In *European Working Session on Learning*, pages 239–250, 1989.
- John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1050–1055, 1996.
- John M Zelle, Raymond J Mooney, and Joshua B Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 343–351, 1994.

Appendices

Appendix A

ASP Code

```
%  
% SemEval 2016 System: Inspire – Interpretable Textual  
% Similarity Alignment based on Answer Set Programming  
%  
% Copyright (C) 2015–2016 Mishal Kazmi  
% Copyright (C) 2015–2016 Peter Schueller  
%  
  
% comfort representation  
word(I,W) :- mword(I,W,L,P,N).  
lword(I,L) :- mword(I,W,L,P,N).  
pos(I,P) :- mword(I,W,L,P,N).  
ner(I,N) :- mword(I,W,L,P,N).  
  
% classify words  
propnoun(X) :- pos(X,("NNP";"NNPS")).  
noun(X) :- pos(X,("NN";"NNS";"PRP";"PRP$";"WP";"WP$")).  
noun(X) :- propnoun(X).  
verb(X) :- pos(X,("VB";"VBD";"VBG";"VBN";"VBP";"VBZ")).  
location(X) :- ner(X,"LOCATION").  
adj(X) :- pos(X,("JJ";"JJR";"JJS")).  
adv(X) :- pos(X,("RB";"RBR";"RBS";"WRB")).  
contentword(X) :- noun(X).  
contentword(X) :- verb(X).  
contentword(X) :- adj(X).  
contentword(X) :- adv(X).  
conjunction(X) :- lword(X,("and";"or";"but";"although";"therefore");
```

```

    "hence";"moreover" )).

% symmetry of similarity
% (we generate it in one direction in Python but we use it in both
% directions in ASP!)
chunksimilarity(C2,C1,S) :- chunksimilarity(C1,C2,S).

% do we have a chunk similarity value?
has_chunksimilarity(C1,C2) :- chunksimilarity(C1,C2,_).

% candidate_pair(SentenceID1, SentenceID2, ChunkID1, ChunkID2)
% pair of chunks in different sentences
chpair(S1,S2,C1,C2) :-
    chunk(C1), chunk(C2), C1 = sc(S1,CIdx1), C2 = sc(S2,CIdx2), S1 != S2.

% has_contentword is true for chunks that contain at least one content word
has_contentword(C) :- contentword(cw(C,_)).

% if require_contentword is true, we care about has_contentword
% if require_contentword is not true, we do not care (it is always true)
has_contentword_or_dontcare(C) :- chunk(C), not require_contentword.
has_contentword_or_dontcare(C) :- has_contentword(C), require_contentword.

% Applying conditions before rules

% condc1(ChunkID/ChunkID): second chunk has conjunction, first one has not
condc1(C1,C2) :- chpair(-,-,C1,C2),
    % C1 has no conjunction and C2 has
    % (we ensure the other case below)
    #count { W1 : conjunction(cw(C1,W1)) } == 0,
    #count { W2 : conjunction(cw(C2,W2)) } >= 1.

% condc2(ChunkID1/ChunkID2)
% A content word in C1 has an antonym in C2
condc2(C1,C2):- chpair(-,-,C1,C2),
    contentword(cw(C1,WI1)), lword(cw(C1,WI1),W1),
    1 <= #count { WI2 : lword(cw(C2,WI2),W2), antonym(W1,W2) }.

% condc3(ChunkID): chunk has numeric entity
condc3(C) :- chunk(C), #count { W : cardinalnumber(cw(C,W)) } >= 1.

```



```

% match(WordID, WordID):
% if two words in different sentences match (ignore case)
match(WI1,WI2) :- chpair(,-,C1,C2), % in chunks in different sentences
    WI1 = cw(C1,W1), WI2 = cw(C2,W2), lword(WI1,W), lword(WI2,W).

% condc4(ChunkID)
% a chunks has a LOCATION entity
condc4(C) :- location(cw(C, _)).

% cond5(ChunkID)
% a chunk has a DATE/TIME entity
condc5(C) :- datetime(cw(C, _)).

% condc6(ChunkID, ChunkID):
% chunks share one content word other than noun
condc6(C1,C2) :- match(cw(C1,W1),cw(C2,W2)), contentword(cw(C1,W1)),
    contentword(cw(C2,W2)), not noun(cw(C1,W1)), not noun(cw(C2,W2)).

% condc7(ChunkID):
% any of the chunks has a conjunction
condc7(C) :- chunk(C), conjunction(cw(C, _)).

% Order of condition application
cond1235(C1,C2) :- condc1(C1,C2). % both directions of condc1
cond1235(C1,C2) :- condc1(C2,C1). % both directions of condc1
cond1235(C1,C2) :- condc2(C1,C2). % both directions of condc2
cond1235(C1,C2) :- condc2(C2,C1). % both directions of condc2
cond1235(C1,C2) :- chpair(,-,C1,C2), condc3(C1).
cond1235(C1,C2) :- chpair(,-,C1,C2), condc3(C2).
cond1235(C1,C2) :- chpair(,-,C1,C2), condc5(C1).
cond1235(C1,C2) :- chpair(,-,C1,C2), condc5(C2).

cond1to5(C1,C2) :- cond1235(C1,C2).
cond1to5(C1,C2) :- chpair(,-,C1,C2), condc4(C1), condc4(C2).
% both chunks have location entities

cond3or7(C) :- condc3(C).
cond3or7(C) :- condc7(C).

```

```

% len(ChunkID,w,Length): number of words in chunk
len(C,w,Length) :- chunk(C), Length = #count{WIdx : word(cw(C,WIdx),-)}.

% Rules
% no1(ChunkID):
% chunk is a single punctuation token
punct(".",",",";","!","?","'","\"").
no1(C) :- chunk(C), pos(cw(C,WID),Pos), punct(Pos), len(C,w,1).

% word_extra_w(C1,C2,CW):
% chunk C1 has word with ID CW that is not in chunk C2
word_extra_w(C1,C2,cw(C1,W1)) :- chpair(-,-,C1,C2),
% word W from C1 is not matched in any word W2 in C2
word(cw(C1,W1),-), 0 = #count { W2: match(cw(C1,W1),cw(C2,W2)) }.

% word_extra(C1,C2):
% chunk C1 has some word that is not in chunk C2
word_extra(C1,C2) :- word_extra_w(C1,C2,-).

% eq1(ChunkID,ChunkID)
% if chunks in different sentences are the same (lowercased)
eq1(C1,C2) :- chpair(-,-,C1,C2), not word_extra(C1,C2),
not word_extra(C2,C1).

% contentword_extra_w(C1,C2,CW):
% chunk C1 has content word with ID CW that is not in chunk C2
contentword_extra_w(C1,C2,W) :- word_extra_w(C1,C2,W), contentword(W).

% contentword_extra(C1,C2):
% chunk C1 has some content word that is not in chunk C2
% in sets: C1 \not\subseteq C2
contentword_extra(C1,C2) :- contentword_extra_w(C1,C2,-).

% contentword_subset(C1,C2)
% chunk C1 is a sub-chunk of chunk C2
% = chunk C1 contains only contentwords from chunk C2
contentword_subset(C1,C2) :- chpair(-,-,C1,C2),
not contentword_extra(C1,C2).

% contentword_match(ChunkID1,ChunkID2)

```

```

% if there is at least one contentword match between chunks
contentword_match(C1,C2) :- chpair( _ , _ , C1,C2), match(cw(C1,WI1),
    cw(C2,WI2)), contentword(cw(C1,WI1)).

% eq2(ChunkID, ChunkID):
% both chunks have same content words
eq2(C1,C2) :- chpair(1,2,C1,C2), contentword_match(C1,C2),
    not contentword_extra(C1,C2), not contentword_extra(C2,C1).

% contentword_extra_notsynonym_w(ChunkID1, ChunkID2, W):
contentword_extra_notsynonym_w(C1,C2,WI1) :-
    contentword_extra_w(C1,C2,WI1), lword(WI1,W1),
    0 = #count { WI2 : lword(cw(C2,WI2),W2), synonym(W1,W2) }.
contentword_extra_notsynonym(C1,C2) :-
    contentword_extra_notsynonym_w(C1,C2, _).

% build transitive reflexive closure over synonyms
synonym(X,Y) :- synonym(Y,X).
synonym(X,Z) :- synonym(X,Y), synonym(Y,Z).

% reflexivity for antonyms
antonym(X,Y) :- antonym(Y,X).

% eq3(ChunkID, ChunkID):
% all content words match using synonym lookup
eq3(C1,C2) :- chpair( _ , _ , C1,C2), not contentword_extra_notsynonym(C1,C2),
    not contentword_extra_notsynonym(C2,C1).

% eq4(ChunkID1, ChunkID2)
% All content words of a chunk match and unmatched content words of other
% chunk are all proper noun type
contentword_extra_notpropernoun_w(C1,C2,WI1) :-
    contentword_extra_w(C1,C2,WI1), not propernoun(WI1).
contentword_extra_notpropernoun(C1,C2) :-
    contentword_extra_notpropernoun_w(C1,C2, _).

% in both directions
eq4(C1,C2):- chpair( _ , _ , C1,C2),
    not cond1to5(C1,C2), % only if none of condition 1 to 5 are fulfilled
    not contentword_extra(C1,C2), not contentword_extra_notpropernoun(C2,C1).

```

```

% both chunks have equal number of content words
eqcontentw(C1,C2):- chpair(1,2,C1,C2),
    0 = #sum{1,W1 : contentword(cw(C1,W1)) ; -1,W2 : contentword(cw(C2,W2))}.

% eq5(ChunkID1, ChunkID2, score)
% Both chunks have equal number of content words and sim Mikolov>0.6
eq5(C1,C2):- eqcontentw(C1,C2),
    not cond1235(C1,C2), % only if none of condition 1,2,3,5 are fulfilled
    chunksimilarity(C1,C2,S), S > 60.

% op1(ChunkID1, ChunkID2):
% A content word in one chunk has an antonym in the other chunk
% (corresponds to cond2)
% not if c3 or c7
op1(C1,C2) :- chpair(-,-,C1,C2), % can be in both directions
    condc2(C1,C2), not cond3or7(C1), not cond3or7(C2).

% sp1(ChunkID A,ChunkID B): chunk A is more specific than chunk B
% chunk A has a conjunction
% and
% chunk A contains all content words of chunk B
sp1(A,B) :- chpair(-,-,A,B),
    % condc1: B has no conjunction, A has at least one conjunction
    condc1(B,A),
    % contentword_subset: B is a sub-chunk of A
    contentword_subset(B,A),
    % both chunks contain at least one content word
    has_contentword_or_dontcare(A), has_contentword_or_dontcare(B).

% sp2(Chunk ID A,Chunk ID B): chunk A is more specific than chunk B
% Chunk A contains all content words of chunk B plus extra content words
% that are not verbs
% Maximum token overlap is selected at spe
sp2candidate(A,B) :- chpair(-,-,A,B),
    % contentword_subset: B is a sub-chunk of A
    contentword_subset(B,A),
    % both chunks have at least one content word
    has_contentword_or_dontcare(A), has_contentword_or_dontcare(B),
    0 == #count { WId : contentword_extra_w(A,B,WId), verb(WId) }.

```

```

% how many tokens in these chunks do overlap?
sp2overlap(A,B,Overlap) :- sp2candidate(A,B),
    Overlap = #count { WAIId : match(cw(A,WAIId),cw(B,WBIId)) }.

% for each chunk A, find the longest overlap
sp2bestoverlap(A,Highest) :- sp2candidate(A,_),
    Highest = #max { Overlap : sp2overlap(A,B,Overlap) }.

% for each chunk A, select one of the longest overlaps
{ sp2choose(A,B) } :- sp2bestoverlap(A,Highest), sp2overlap(A,B,Highest).

% choose exactly one for each A
:- sp2candidate(A,_), not 1 = #count { A: sp2choose(A,B) }.

sp2(A,B):- sp2choose(A,B).

% sp3(Chunk ID A,Chunk ID B): chunk A is more specific than chunk B
% Chunk A and B contain only one noun each and hypernym
% determines which is more specific
sp3onenouneach(C1,C2):- chpair(-,-,C1,C2),
    1 = #count{W1 : noun(cw(C1,W1))},
    1 = #count{W2 : noun(cw(C2,W2))}.
sp3(C2,C1) :-
    sp3onenouneach(C1,C2),
    noun(cw(C1,W1)), noun(cw(C2,W2)),
    lword(cw(C1,W1),W1String), lword(cw(C2,W2),W2String),

% hypernym(X,Y): Y is more specific than X
hypernym(W1String,W2String).

% si1(ChunkID1,ChunkID2):
% Only unmatched content word in each chunk is a cardinal number type
silcandidate(C1,C2) :- chpair(1,2,C1,C2),
    % only one extra content word
    1 = #count { W1 : contentword_extra_w(C1,C2,W1) },
    % only one extra content word
    1 = #count { W2 : contentword_extra_w(C2,C1,W2) }.
sil(C1,C2) :- silcandidate(C1,C2),

```

```

% those are cardinals
contentword_extra_w(C1,C2,W1), cardinalnumber(W1),
% those are cardinals
contentword_extra_w(C2,C1,W2), cardinalnumber(W2).

% si2: both chunks have DATE/TIME entities
si2(C1,C2) :- chpair(1,2,C1,C2), condc5(C1), condc5(C2).

% si3(ChunkID1, ChunkID2):
% Each chunk has a token of LOCATION type
si3(C1,C2):- chpair(1,2,C1,C2), condc4(C1), condc4(C2).

% si4(ChunkID1, ChunkID2):
% Both chunks share atleast one noun:
% if sim Mikolov >= 0.4 then score = 3 otherwise 2
si4sim(C1,C2,S) :- chpair(1,2,C1,C2), match(cw(C1,W1),cw(C2,W2)),
noun(cw(C1,W1)), noun(cw(C2,W2)), chunksimilarity(C1,C2,S).

% si5sim(ChunkID1, ChunkID2, Similarity):
% if condition 6 not satisfied
% score = 4 if sim Mikolov in [0.7, -1.0]
% score = 3 if sim Mikolov in [0.65, -0.7)
% score = 2 if sim Mikolov in [0.60, -0.65)
si5sim(C1,C2,S):- chpair(1,2,C1,C2),
has_contentword_or_dontcare(C1), has_contentword_or_dontcare(C2),
not condc6(C1,C2), chunksimilarity(C1,C2,S).

% re1sim(ChunkID1, ChunkID2, Similarity):
% if both chunks share atleast one content word other than noun
re1sim(C1,C2,S):- chpair(1,2,C1,C2),
has_contentword_or_dontcare(C1), has_contentword_or_dontcare(C2),
condc6(C1,C2), chunksimilarity(C1,C2,S).

% get final alignment from last step
% final(Chunk ID, Relation, Score, Chunk ID, Rule Causing Alignment,
% Mikolov-Score)

% define similarity for all candidate pairs
similarity_or_none(C1,C2,Mikolov) :-
chpair(1,2,C1,C2), chunksimilarity(C1,C2,Mikolov).

```

```

similarity_or_none(C1,C2,null) :-
    chpair(1,2,C1,C2), not has_chunksimilarity(C1,C2).

% a final chunk alignment is
final(C1,Rel,S,C2,Step,Mikolov) :-
    % caused by aligning chunks that are not already aligned at that step
    chalign(C1,Rel,S,C2,Step), not aligned(C1,Step), not aligned(C2,Step),
    % we also use this step
    usedStep(Step),
    % and uses similarity if it exists
    similarity_or_none(C1,C2,Mikolov).

% has a chunk been aligned in a certain step? then it is already aligned
% in next step
aligned(C,NextStep) :- chalign(C,-,-,-,Step), nextStep(Step,NextStep).
aligned(C,NextStep) :- chalign(-,-,-,C,Step), nextStep(Step,NextStep).

% what is aligned stays aligned
aligned(C,NextStep) :- aligned(C,Step), nextStep(Step,NextStep).
chalign(C1,R,S,C2,NextStep) :-
    chalign(C1,R,S,C2,Step), nextStep(Step,NextStep).

usedStep(X) :- nextStep(X,_).
usedStep(X) :- nextStep(-,X).

% define NOALI alignments
chalign(C,"NOALI",0,null,noalic) :- chunk(C), C = sc(1,C1),
    not aligned(C,noalic), no1(C).
chalign(null,"NOALI",0,C,noalic) :- chunk(C), C = sc(2,C1),
    not aligned(C,noalic), no1(C).

% define EQUI alignments
chalign(C1,"EQUI",5,C2,eqi1) :- chpair(1,2,C1,C2),
    not aligned(C1,eqi1), not aligned(C2,eqi1), eq1(C1,C2).
chalign(C1,"EQUI",5,C2,eqi2) :- chpair(1,2,C1,C2),
    not aligned(C1,eqi2), not aligned(C2,eqi2), eq2(C1,C2).
chalign(C1,"EQUI",5,C2,eqi3) :- chpair(1,2,C1,C2),
    not aligned(C1,eqi3), not aligned(C2,eqi3), eq3(C1,C2).
chalign(C1,"EQUI",5,C2,eqi4) :- chpair(1,2,C1,C2),
    not aligned(C1,eqi4), not aligned(C2,eqi4), eq4(C1,C2).

```

```

chalign(C1,"EQUI",5,C2,equi5) :- chpair(1,2,C1,C2),
    not aligned(C1,equi5), not aligned(C2,equi5), eq5(C1,C2).

% define OPPO alignements
chalign(C1,"OPPO",4,C2,oppo) :- chpair(1,2,C1,C2),
    not aligned(C1,oppo), not aligned(C2,oppo), op1(C1,C2).

% define SPE1/SPE2 alignements
% sp1/sp2/sp3 (ChunkID A, ChunkID B): chunk A is more specific than chunk B
chalign(C1,"SPE1",4,C2,sp1) :- chpair(1,2,C1,C2),
    not aligned(C1,sp1), not aligned(C2,sp1), sp1(C1,C2).
chalign(C1,"SPE2",4,C2,sp1) :- chpair(1,2,C1,C2),
    not aligned(C1,sp1), not aligned(C2,sp1), sp1(C2,C1).

chalign(C1,"SPE1",4,C2,sp2) :- chpair(1,2,C1,C2),
    not aligned(C1,sp2), not aligned(C2,sp2), sp2(C1,C2).
chalign(C1,"SPE2",4,C2,sp2) :- chpair(1,2,C1,C2),
    not aligned(C1,sp2), not aligned(C2,sp2), sp2(C2,C1).

chalign(C1,"SPE1",4,C2,sp3) :- chpair(1,2,C1,C2),
    not aligned(C1,sp3), not aligned(C2,sp3), sp3(C1,C2).
chalign(C1,"SPE2",4,C2,sp3) :- chpair(1,2,C1,C2),
    not aligned(C1,sp3), not aligned(C2,sp3), sp3(C2,C1).

% define SIMI alignements
chalign(C1,"SIMI",3,C2,simi1) :- chpair(.,.,C1,C2),
    not aligned(C1,simi1), not aligned(C2,simi1), si1(C1,C2).
chalign(C1,"SIMI",3,C2,simi2) :- chpair(.,.,C1,C2),
    not aligned(C1,simi2), not aligned(C2,simi2), si2(C1,C2).
chalign(C1,"SIMI",3,C2,simi3) :- chpair(.,.,C1,C2),
    not aligned(C1,simi3), not aligned(C2,simi3), si3(C1,C2).
chalign(C1,"SIMI",3,C2,simi4) :- chpair(.,.,C1,C2),
    not aligned(C1,simi4), not aligned(C2,simi4), si4sim(C1,C2,S), S >= 40.
chalign(C1,"SIMI",2,C2,simi4) :- chpair(.,.,C1,C2),
    not aligned(C1,simi4), not aligned(C2,simi4), si4sim(C1,C2,S), S < 40.
chalign(C1,"SIMI",4,C2,simi5) :- chpair(.,.,C1,C2),
    not aligned(C1,simi5), not aligned(C2,simi5), si5sim(C1,C2,S), 70 <= S.
chalign(C1,"SIMI",3,C2,simi5) :- chpair(.,.,C1,C2),
    not aligned(C1,simi5), not aligned(C2,simi5), si5sim(C1,C2,S),
    65 <= S, S < 70.

```



```

chalign(C1,"SIMI",2,C2,simi5) :- chpair(_,_ ,C1,C2),
  not aligned(C1,simi5), not aligned(C2,simi5), si5sim(C1,C2,S),
  60 <= S, S < 65.
chalign(C1,"SIMI",1,C2,simi5) :- chpair(_,_ ,C1,C2),
  not aligned(C1,simi5), not aligned(C2,simi5), si5sim(C1,C2,S),
  55 <= S, S < 60.

% define REL alignements
chalign(C1,"REL",4,C2,rel1) :- chpair(_,_ ,C1,C2),
  not aligned(C1,rel1), not aligned(C2,rel1), relsim(C1,C2,S), 50 <= S.
chalign(C1,"REL",3,C2,rel1) :- chpair(_,_ ,C1,C2),
  not aligned(C1,rel1), not aligned(C2,rel1), relsim(C1,C2,S),
  40 <= S, S < 50.
chalign(C1,"REL",2,C2,rel1) :- chpair(_,_ ,C1,C2),
  not aligned(C1,rel1), not aligned(C2,rel1), relsim(C1,C2,S), S < 40.

% hide everything
#show.

% show what we need for extracting alignments
#show final/6.
#show word/2.

```